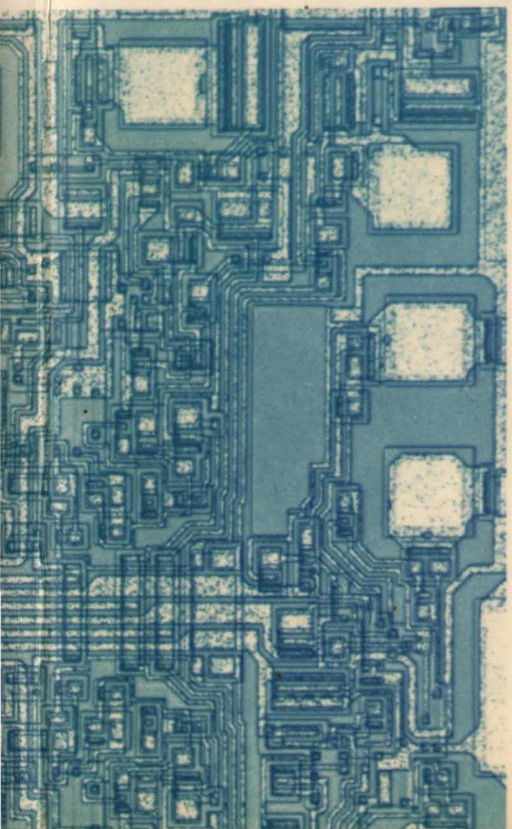
E. Flögel

Programmieren in Maschinensprache mit dem 6502



Apple · ATARI · AIM · CBM · PET VC–20 · Ohio ·

Einführung und Programme

ISBN 3-921682 - 61–4

2. Auflage Herbst 1981

Gedruckt in der Bundesrepublik Deutschland – Printed in West-Germany – Imprime'en RFA.

Vorwort

Im Bereich der Personal-Computer ist der 6502 immer noch der am weitesten verbreitete Prozessor. Dies wird sich auch in der nächsten Zeit nicht ändern, denn Computer, die jetzt neu auf den Markt kommen, verwenden ihn ebenfalls.

Jeder ernsthafte Benutzer eines Computers wird sich nicht nur mit der Kenntniss einer höheren Programmiersprache begnügen, sondern wird versuchen, auch die Maschinensprache zu verstehen und zu beherrschen. Dies ist vor allen dann notwendig, wenn an den Rechner Peripheriegeräte wie Drucker, Wandler, Steuerungen angeschlossen werden, und somit eine schnelle Datenübertragung zwischen Umwelt und Rechner stattfinden soll.

Dazu soll ihm das vorliegende Buch, Programmieren in Maschinensprache 6502, verhelfen. Es ist die völlige Neubearbeitung der bereits früher erschienenen Ausgabe gleichen Titels.

In dieser Ausgabe sind sehr viele Beispiele hineingenommen worden, denn damit lässt sich eine Computersprache leichter erlernen. Ferner werden die wichtigsten 6502 Systeme mit ihren Eigenheiten bei der Assembler-Programmierung vorgestellt.

Dadurch ist erstens ein Vergleich der Systeme möglich, und zweitens können die Programmbeispiele an jedem dieser Rechner ausgeführt werden. Das Lernen ist somit kein trockener Vorgang, sondern eine reizvolle Beschäftigung mit den Computern.

Dazu wünschen wir den Lesern viel Spaß und Erfolg !

Holzkirchen,
Herbst 1981

Die Verfasser
E. Flögel W. Hofacker

Inhaltsverzeichnis

Einführung

1. Einführung

Das Rechnen mit Zahlen war immer ein unbeliebter Vorgang. Deshalb wurde auch schon im Altertum versucht, dieses Rechnen maschinell durchzuführen. Dazu ist es aber notwendig, eine Zahl physikalisch darzustellen. Eine dieser Möglichkeiten bestand darin, eine Zahl einer Folge von Zähnen eines Zahnrades zuzuordnen. Nach diesem Prinzip entstanden sehr leistungsfähige Maschinen für die 4 Grundrechnungsarten.

Beim Übergang von den mechanischen zu den elektronischen Rechenmaschinen mußte nun für die Zahlen eine Darstellung durch Spannungspegel gesucht werden. Die einfachste und sicherste Methode war hier, die Zahl als eine Folge von hohen und niederen Spannungspegeln darzustellen. So entstand als kleinste Einheit einer Zahl das Bit. Damit lassen sich zwei Zustände, d. h. zwei Zahlen Null und Eins darstellen.

1.1 Zahlendarstellung

Während wir im dezimalen Zahlensystem für eine Stelle die Zahlen 0 bis 9 zur Verfügung haben, haben wir im binären Zahlensystem nur die Zahlen 0 und 1. Mit 2 Stellen, d. h. mit 2 Bit können schon 4 Zahlen (Abbildung 1.1), mit 3 Stellen 8 Zahlen dargestellt werden.

2	1	
0	0	0
0	1	1
1	0	2
1	1	3
Binär		Dezimal

Abbildung 1.1

Der Zahlenumfang beträgt also mit N Bit 2^N Zahlen, er erfasst somit die Zahlen von 0 bis $2^N - 1$.

Bei der physikalischen Realisierung entsprechen den Bit im binären Zahlensystem elektrische Leitungen im Mikroprozessor. Bei den ersten Mikroprozessoren (INTEL 4004) betrug die Stellenanzahl 4. Binäre Zahlen mit 4 Stellen konnten somit verarbeitet werden. Die nächste Entwicklungsstufe brachte dann die heute am weitesten verbreiteten Mikroprozessoren mit 8 Bit. Zu diesen gehört auch die 6502 CPU (Central Prozessing Unit), mit deren Programmierung wir uns hier beschäftigen wollen.

Neuere Entwicklungen sind jetzt 16 und 32 Bit Prozessoren.

Eine Binärzahl mit 8 Stellen bezeichnet man auch als Byte. Mit einem Byte können die Zahlen von 0 bis 255 dargestellt werden (Abbildung 1.2).

N =	8	7	6	5	4	3	2	1	
	0	0	0	0	0	0	0	0	0
	1	1	1	1	1	1	1	1	255

Abbildung 1.2

Der Zahl 198 entspricht die folgende Binärzahl:

$$198 \triangleq 1\ 1\ 0\ 0\ 0\ 1\ 1\ 0$$

Die einzelnen Bit eines Bytes haben die folgende Wertigkeit:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Vielfach werden die Stellen eines Byte auch durch den Exponenten der Zweierpotenz gekennzeichnet. Bit 0 ist das am weitesten rechts

stehende, Bit 7 ist das am weitesten links stehende Bit. Dabei ist Bit 0 das Bit mit der geringsten Wertigkeit, und daher auch oft mit LSB (Least Significant Bit) bezeichnet. Demgegenüber wird Bit 7, mit der höchsten Wertigkeit, mit MSB (Most Significant Bit) bezeichnet.

2^n	n
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024	10
2048	11
4096	12
8192	13
16384	14
32768	15
65536	16

1.3 Zweierpotenzen von 0 bis 16

Für dieses Bitmuster muß nun noch eine einfache Darstellung gesucht werden, mit der man dies auch sprachlich angeben kann.

Dazu teilt man die 8-bit in zwei 4-bit Worte auf. Die 16 möglichen Zahlen werden nun in der folgenden Weise bezeichnet (Abbildung 1.4).

Für die Zahlen 0 bis 9 werden die dezimalen Bezeichnungen, für die Zahlen 10 bis 15 die Buchstaben A bis F verwendet.

Für diese Darstellung hat sich im angelsächsischen Sprachgebiet die Bezeichnung Hexadecimalsystem oder kurz HEX-System eingeführt.

Die richtige Bezeichnung ist aber Sedezimalsystem. Wenn wir hier jedoch bei der Bezeichnung Hexadezimal bleiben so liegt dies daran, daß diese Bezeichnung allgemein üblich ist.

0 0 0 0	0
0 0 0 1	1
0 0 1 0	2
0 0 1 1	3
0 1 0 0	4
0 1 0 1	5
0 1 1 0	6
0 1 1 1	7
1 0 0 0	8
1 0 0 1	9
1 0 1 0	A
1 0 1 1	B
1 1 0 0	C
1 1 0 1	D
1 1 1 0	E
1 1 1 1	F

1.4 Sedezimalsystem (Hexadezimalsystem)

Die Zahl 198 mit dem Bitmuster 1 1 0 0 0 1 1 0 entspricht demnach im Hexadezimalsystem die Bezeichnung C6.

198 ≙ 1 1 0 0 0 1 1 0 ≙ C6

Im folgenden werden nun Zahlen in allen 3 Zahlensystemen auftreten. Um nun Verwechslungen zu vermeiden, werden folgende Vereinbarungen getroffen:

Eine Hexadezimalzahl wird durch ein vorgestelltes Dollarzeichen gekennzeichnet, z. B. $C6 , $1000 usw.

Eine binäre Zeichenfolge wird durch ein vorgestelltes Prozentzeichen gekennzeichnet, z. B. %0 0 1 0 1 1 0 0.

Eine Zahl ohne weitere Angabe ist eine Dezimalzahl.

198 ≙ % 1 1 0 0 0 1 1 0 ≙ $C6

Ein Bitmuster von 8 bit kann auf verschiedene Weise interpretiert werden. Wir haben bis jetzt gesehen, daß damit 256 Zahlen dargestellt werden können. Dieses Bitmuster kann aber auch ein Zeichen oder ein Befehl für den Prozessor sein. Der eigentliche Anfang der programmierbaren Rechenautomaten begann mit der Idee, Zahlen und Befehle durch Bitmuster darzustellen und sie in einem gemeinsamen Speicher abzulegen.

1.2 Vorzeichenbehaftete Zahlen

Sollen durch das Bitmuster positive und negative Zahlen bezeichnet werden, so muß ein Bit als Vorzeichenbit interpretiert werden. Dazu wird Bit 7 benutzt. Mit Bit 7 = 0 ist die Zahl positiv, mit Bit 7 = 1 ist die Zahl negativ.

Somit ist $00 = 0
$01 = 1
.
.
.
$7F = 127

Für negative Zahlen wird eine andere Darstellung verwendet, das sogenannte 2-er Complement. Complementieren einer Binärzahl bedeutet vertauschen von Null und Eins. Bei der Bildung des 2-er Complements wird zu dieser Vertauschung noch eine Eins dazuaddiert.
Beispiel: Bestimmung der Zahl –1.

	+1	≙	% 0 0 0 0 0 0 0 1		
Complement		≙	% 1 1 1 1 1 1 1 0		
Eins dazuaddiert			+% 0 0 0 0 0 0 0 1		
ergibt	–1	≙	% 1 1 1 1 1 1 1 1	≙	$FF

Die Zahl –1 wird also durch % 1 1 1 1 1 1 1 1 oder $FF gekennzeichnet.

Die Abbildung 1.5 zeigt die Zahlengerade mit positiven und negativen Binärzahlen.

$80	$81	…	$FE	$FF	$00	$01	$02	…	$7E	$7F
−128	−127	…	−2	−1	0	1	2	…	126	127

1.5 Zahlengerade

1.3 Rechenregeln und logische Verknüpfungen

Im Beispiel für die Bildung einer negativen Zahl wurde die Addition zweier Binärzahlen ausgeführt. Hierfür gelten folgende Rechenregeln:

```
  0      0      1      1
 +0     +1     +0     +1
----   ----   ----   ----
  0      1      1     1|0
```

Beispiel: 132 + 20 = 152

```
 132  ≙  % 1 0 0 0 0 1 0 0
 +20  ≙  % 0 0 0 1 0 1 0 0
---------------------------
 152  ≙  % 1 0 0 1 1 0 0 0
```

Eine Subtraktion wird durch Addition der negativen Zahl durchgeführt.

Beispiel: 98 − 20 = 78

```
               20  ≙  % 0 0 0 1 0 1 0 0
Complement         ≙  % 1 1 1 0 1 0 1 1
               +1     % 0 0 0 0 0 0 0 1
-----------------------------------------
              −20  ≙  % 1 1 1 0 1 1 0 0

               98  ≙  % 0 1 1 0 0 0 1 0
          + (−20)  ≙  % 1 1 1 0 1 1 0 0
-----------------------------------------
             + 78  ≙ % 1|0 1 0 0 1 1 1 0
```

Der entstehende Übertrag weist darauf hin, daß die Subtrakiton ohne "Borgen" ausgeführt wurde und der Rest eine positive ganze Zahl ist.

Beispiel: 20 – 98 = –78

98	≙	% 0 1 1 0 0 0 1 0
Complement	≙	% 1 0 0 1 1 1 0 1
+ 1	≙	% 0 0 0 0 0 0 0 1
– 98	≙	% 1 0 0 1 1 1 1 0

20	≙	% 0 0 0 1 0 1 0 0
+ (– 98)	≙	% 1 0 0 1 1 1 1 0
– 78	≙	% 1 0 1 1 0 0 1 0

Der Betrag der negativen Zahl – 78 wird durch Rückwandlung in eine positive Zahl gebildet

– 78	≙	% 1 0 1 1 0 0 1 0
+ (– 1)	≙	% 1 1 1 1 1 1 1 1
		% 1 | 1 0 1 1 0 0 0 1
Complement |78|	≙	% 0 1 0 0 1 1 1 0

1.4 Logische Funktionen

Folgende logische Funktionen können auf binäre Zahlen angewendet werden:

1. Negation
   Die negierte Zahl wird durch Überstreichen gekennzeichnet.

   $B = 0 ; \overline{B} = 1$ und

   $B = 1 ; \overline{B} = 0$

2. Die UND-Funktion

A & B = C

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Diese Funktion wird zum Ausblenden bestimmter Teile eines Bitmusters benötigt.

B	=	% 1 0 1 1 0 1 0 1
A	=	% 0 1 1 1 1 0 0 0
B & A	=	% 0 0 1 1 0 0 0 0

Das Bitmuster B wird nur an diesen Stellen ins Ergebnis übernommen, in denen im Ausblendmuster A eine 1 steht.

3. Die ODER-Funktion

A V B = C

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Mit der ODER-Funktion können bestimmte Bit im Ergebnis zwangsweise zu 1 gemacht werden.

B	=	% 0 1 0 1 1 0 0 1
A	=	% 1 0 0 0 0 0 1 1
B V A	=	% 1 1 0 1 1 0 1 1

4. Die EXCLUSIV ODER-Funktion

$A \wedge B = C$

A	B	C
0	0	0
1	0	1
0	1	1
1	1	0

Mit der EXCLUSIV ODER-Funktion kann das Complement eines Bitmusters gebildet werden.

B	=	% 1 0 1 1 0 1 0 1
A	=	% 1 1 1 1 1 1 1 1
$B \wedge A$	=	% 0 1 0 0 1 0 1 0

4		3		2		1	
HEX	DEZ	HEX	DEZ	HEX	DEZ	HEX	DEZ
0	0	0	0	0	0	0	0
1	4,096	1	256	1	16	1	1
2	8,192	2	512	2	32	2	2
3	12,288	3	768	3	48	3	3
4	16,384	4	1,024	4	64	4	4
5	20,480	5	1,280	5	80	5	5
6	24,576	6	1,536	6	96	6	6
7	28,672	7	1,792	7	112	7	7
8	32,768	8	2,048	8	128	8	8
9	36,864	9	2,304	9	144	9	9
A	40,960	A	2,560	A	160	A	10
B	45,056	B	2,816	B	176	B	11
C	49,152	C	3,072	C	192	C	12
D	53,248	D	3,328	D	208	D	13
E	57,344	E	3,584	E	224	E	14
F	61,440	F	3,840	F	240	F	15

1.6 Tabelle zur Umwandlung von Hexadezimalzahlen in Dezimalzahlen

In Abbildung 1.6 ist eine Tabelle zur Umwandlung von Hexadezimalzahlen in Dezimalzahlen angegeben. Für die 4-stellige Hexzahl $ABCD erhält man als Dezimalzahl

$ABCD = 40960 + 2816 + 192 + 13 = 43981

Die Umwandlung einer Dezimalzahl in eine Hexadezimalzahl wird durch Subtraktion des nächst kleineren Dezimalwertes einer Hexadezimalstelle erreicht.

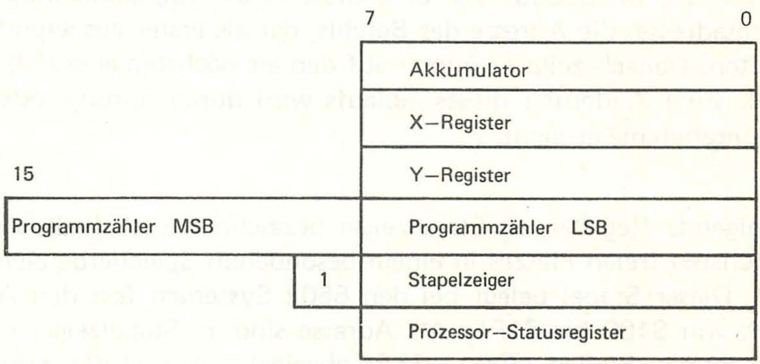
Beispiel:
Umwandlung von 33333 in Hexadezimalzahl

```
  33333                          33333 = $8235
- 32768  = $8000
    565
  - 512  = $ 200
     53
   - 48  = $  30
      5
    - 5  = $   5
      0
```

Aufbau der CPU 6502

2. Aufbau der CPU 6502

In der Abbildung 2.1 ist die CPU 6502 in einer Form dargestellt, die alle dem Programmierer zugänglichen Teile zeigt. Mit diesen internen Registern wird der Datenfluß zwischen den Teilen eines Mikroprozessorsystems geregelt.



2.1 Logische Struktur der CPU 6502

Das zentrale Register ist der Akkumulator. Bei der Ausführung einer arithmetischen Operation ist ein Operand im Akkumulator, der andere im Speicher. Das Ergebnis wird im Akkumulator abgelegt und kann von dort in den Speicher zurückgeschrieben werden. Der Operand, der sich vorher im Akkumulator befand wird also überschrieben und geht somit verloren.

Sollen Daten im Speicher verschoben werden, so müssen sie erst durch einen LADE-Befehl in den Akkumulator übernommen und dann durch

einen SPEICHER-Befehl zurückgeschrieben werden.

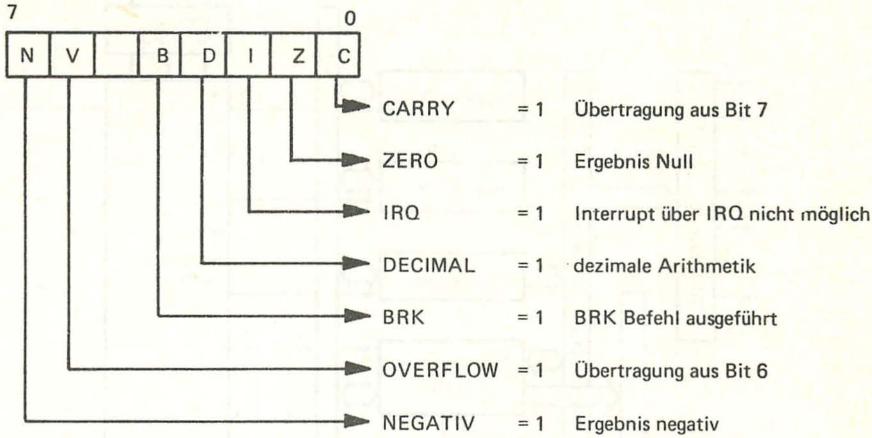
Die beiden folgenden Register, mit X-, bzw Y-Register bezeichnet, können ebenfalls Daten aus dem Speicher übernehmen und wieder zurückschreiben. Die Ausführung von arithmetischen Befehlen ist allerdings eingeschränkt. Der Inhalt dieser Register kann jeweils nur um Eins erhöht oder erniedrigt werden. Damit sind sie aber besonders als Zählregister (Indexregister) zum Abzählen von Programmschleifen geeignet. Wie wir bei den Adressierungsarten noch sehen werden, kann mit dem Inhalt des X- oder Y-Registers eine programmierte Adresse verändert werden. Auf diese 3 Register, Akkumulator, X- und Y-Register kann der Programmierer sehr leicht zugreifen.

Weit weniger zugänglich sind die folgenden Register. Der Programmzähler belegt zwei 8-bit Register. Somit reicht also der Bereich der ansprechbaren Adressen von $0000 bis $FFFF oder dezimal ausgedrückt von 0 bis 65535. Vor dem Start eines Programms muß er die Anfangsadresse, die Adresse des Befehls, der als erster ausgeführt wird, enthalten. Danach zeigt er immer auf den als nächsten auszuführenden Befehl. Eine Änderung dieses Ablaufs wird durch Sprung- oder Verzweigungsbefehle erreicht.

Das folgende Register, als Stapelzeiger bezeichnet, enthält die Adresse des nächsten freien Platzes in einem besonderem Speicherbereich, dem Stapel. Dieser Stapel belegt bei den 6502 Systemen fest den Adressbereich von $100 bis $1FF. Als Adresse sind im Stapelzeiger nur die beiden letzten Stellen ($00 – $FF) abgelegt, die 1 in der vordersten Stelle wird vom Prozessor automatisch hinzugeführt. Während eines Programmablaufs werden bei Unterprogrammaufrufen oder bei Interruptanforderungen die augenblicklichen Adressen des Programmzählers aufbewahrt. Bei einem Interrupt auch noch zusätzlich der Inhalt des folgenden Registers, dem Prozessorstatus-Register. In diesem wird in einzelnen Bits der augenblickliche Zustand des Prozessors gespeichert. Ihre Zuordnung ist in Abbildung 2.2 dargestellt.

Die Bit 0, 1, 6 und 7 werden durch Rechenoperationen beeinflußt. Bit 0 (CARRY) ist 1, wenn bei der Ausführung des Befehls ein Übertrag aus Bit 7 des Akkumulators eintrat. Bit 1 (ZERO) ist 1, wenn das Ergebnis der Rechenoperation Null ist. Bit 6 (OVERFLOW) wird 1,

wenn ein Übertrag im Akkumulator von Bit 6 nach Bit 7 stattfindet, und schließlich wird Bit 7 gleich 1, wenn das Ergebnis eine negative Zahl darstellt.



2.2 Belegung der Bit im Statusregister

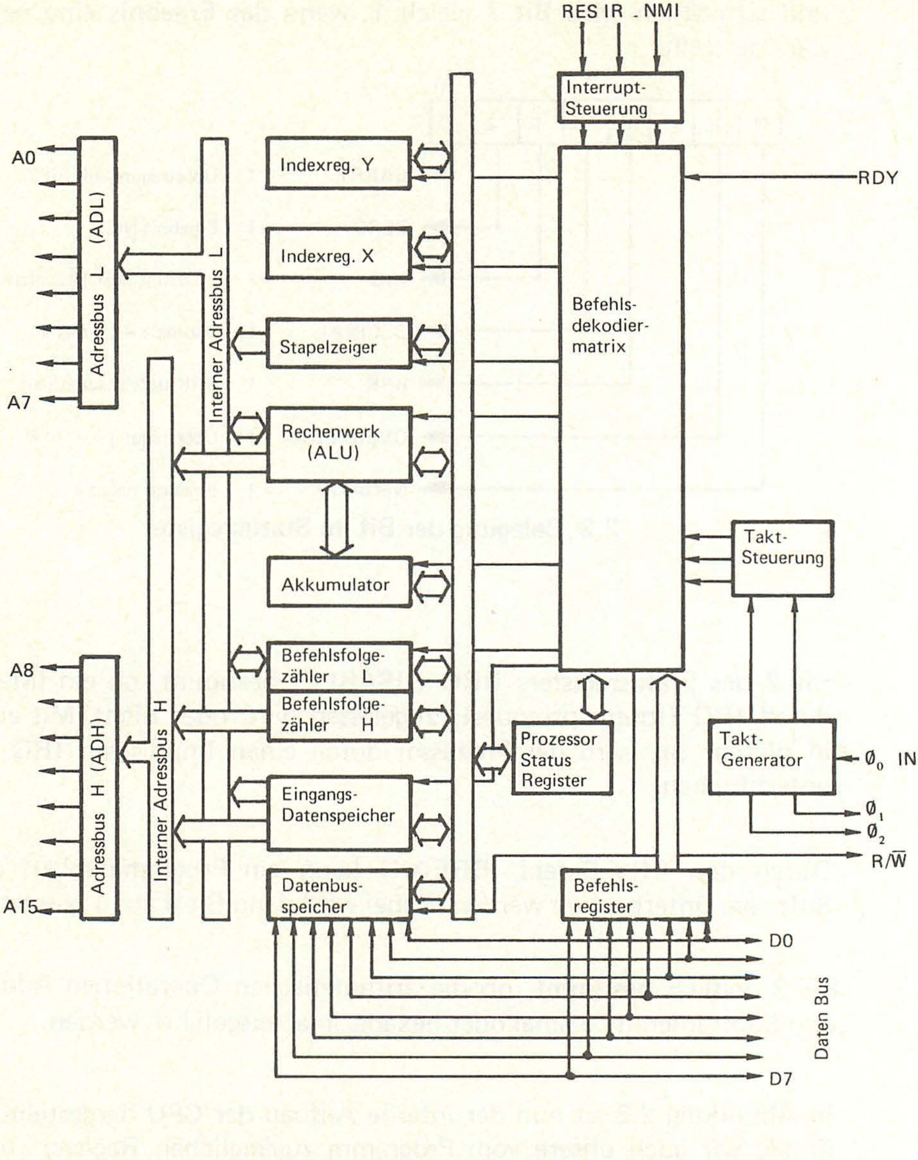
Bit 2 des Statusregisters (IRQ DISABLE) bestimmt, ob ein Interrupt durch IRQ (Interruptrequest) zugelassen wird oder nicht. Mit einer 1 in diesem Bit wird der Prozessor durch einen Impuls auf IRQ nicht unterbrochen.

Durch den BRK-Befehl (BREAK) kann ein Programmablauf durch Software unterbochen werden. Dabei wird dann Bit 4 auf 1 gesetzt.

Bit 3 endlich bestimmt, ob die arithmetischen Operationen Addieren und Subtrahieren dezimal oder hexadezimal ausgeführt werden.

In Abbildung 2.3 ist nun der interne Aufbau der CPU dargestellt. Hier finden wir auch unsere vom Programm zugänglichen Register (Indexregister, Akkumulator, Statusregister, Programmzähler) wieder. Daneben sind aber auch noch weitere Komponenten vorhanden. Das zentrale Rechenwerk ist die ALU (Arithmetik Logical Unit).

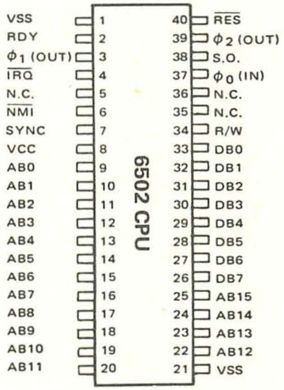
Diese führt alle Rechenoperationen aus.



2.3 Interne Struktur der CPU 6502

Ein in die CPU eingelesenes Bitmuster wird gespeichert, und wenn es sich um einen Befehl handelt vom Befehlsdecoder entschlüsselt. Über einen internen Datenbus werden die Bitmuster dem jeweiligen Register zugeordnet. Handelt es sich um Adressen werden sie über interne Adressleitungen auf die Adressausgänge der CPU durchgeschaltet.

Die CPU selbst benötigt zum Betrieb nur einen externen Taktgenerator. Intern werden dann alle notwendigen Signale erzeugt. Abbildung 2.4 zeigt die Pinbelegung eines 6502 im 40 poligen Gehäuse.



2.4 6502 Pinbelegung

NOTIZEN

Befehlsarten

3. Befehlsarten

Der Befehlsvorrat der 6502 CPU wird zum besseren Verständnis in 8 Gruppen aufgeteilt. Diese Einteilung wird so getroffen, daß Befehle, die gleiche Funktionen haben, in einer Gruppe zusammengefasst sind. Wir beginnen mit den Transportbefehlen. Mit diesen Befehlen wird ein Bitmuster z. B. vom Akkumulator in den Speicher oder in ein Indexregister transportiert.

Eine kurze Bemerkung zur Bezeichnung der Befehle. Im Rechner selbst wird ein Befehl durch ein Bitmuster, bestehend aus 0 oder 1 dargestellt. Dieses Bitmuster kann auch durch eine Hexadezimalzahl angegeben werden. Wir verwenden aber hier eine abgekürzte Schreibweise, die wir auch später als Assemblerschreibweise kennenlernen werden. Nun zu den Befehlsgruppen.

3.1 Transportbefehle

LDA (Load Accu)	Laden den Akkumulator mit dem Inhalt einer Speicherzelle.
LDX LDY	Lade das X-, bzw. das Y-Register mit dem Inhalt einer Speicherzelle.
STA (Store Accu)	Speichere den Inhalt des Akkumulators in eine Speicherzelle.
STX STY	Speichere den Inhalt des X-, bzw. Y-Registers in eine Speicherzelle.
TAX TAY	Transportiere den Inhalt des Akkumulators in das X-, bzw. Y-Register.
TXA TYA	Transprotiere den Inhalt des X-, bzw. Y-Registers in den Akkumulator.

TSX		Transportiere den Stapelzeiger in das X-Register.
TXS		Transportiere das X-Register in den Stapelzeiger.
PLA PLP	(Pull Accumulator from Stack) (Pull Prozessor Status from Stack)	Der Inhalt der Speicherzelle im Stapelspeicher, auf die der Stapelzeiger hinweist, wird in den Akkumulator, bzw. in das Prozessorstatus-Register übernommen. Der Inhalt des Stapelzeigers wird um Eins erhöht.
PHA PHP	(Push Accumulator on Stack) (Push Prozessor Status on Stack)	Der Inhalt des Akkumulators bzw. des Prozessorstatus-Registers wird in die Speicherzelle des Stapels geschrieben, auf die der Stapelzeiger hinweist. Anschließend wird der Inhalt des Stapelzeigers um Eins erhöht.

3.2 Arithmetische Befehle

Die 6502 CPU besitzt nur Befehle zum Addieren und Subtrahieren, keine Multiplikations- oder Divisionsbefehle.

ADC SBC	(Add with Carry) (Subract with Carry)	Addiere den Inhalt einer Speicherzelle zum Inhalt des Akkumulators. Erhöhe dieses Ergebnis um 1, wenn das Übertragsbit = 1 ist. Subtrahiere den Inhalt einer Speicherzelle vom Inhalt des Akkumulators. Erniedrige dieses Ergebnis um 1, wenn das Übertragsbit 1 ist.
INC INX INY	(Increment)	Erhöhe den Inhalt einer Speicherzelle des X-, bzw. Y-Registers um 1.
DEC DEX DEY	(Decrement)	Erniedrige den Inhalt einer Speicherzelle, des X-, bzw. Y-Registers um 1.

3.3 Logische Befehle

Mit dieser Befehlsgruppe werden der Inhalt des Akkumulators und der Inhalt einer Speicherzelle durch logische Operationen verknüpft. Das Ergebniss ist im Akkumulator gespeichert.

AND	UND-Funktion
ORA	ODER-Funktion
EOR	Exclusiv ODER-Funktion

3.4 Vergleichsbefehle

Mit diesen Vergleichsbefehlen wird der Inhalt des Akkumulators oder der Indexregister mit dem Inhalt einer Speicherzelle verglichen. Das Ergebnis dieses Vergleichs setzt die entsprechenden Bit im Prozessor-status-Register. Der Inhalt des Akkumulators wird nicht verändert.

CMP (Compare) — Vergleiche den Inhalt einer Speicherzelle
CPX — mit dem Akkumulator, dem X-, bzw.
CPY — Y-Register.

In Abbildung 3.1 wird dargestellt, in welcher Weise die Bit im Statusregister gesetzt werden. Eine besondere Beachtung bedarf hier das N Bit des Statusregisters. Dieses Bit wird durch die Vergleiche so gesetzt, als ob die Zahlen im 2-er Complement dargestellt sind, d. h. daß der Zahlenbereich nicht von 0 bis 255, sondern von – 128 bis + 127 reicht.

Vergleich	N	Z	C
A, X, Y 〈 M	1*	0	0
A, X, Y = M	0	1	1
A, X, Y 〉 M	0*	0	1

*Vergleich im 2-er Complement

3.1 Setzen der Bit im Statusregister durch Vergleichsbefehle

Werden folgende zwei Zahlen miteinander vergleichen, 〈 A 〉 = $00,

< M > = $FF so ist für das ZERO- und CARRY-Bit der Inhalt des Akkumulators kleiner als der Inhalt der Speicherzelle (A < M) und somit sind ZERO- und CARRY-Bit Null (Z = 0, C = 0). Bei der Darstellung im 2-er Complement bedeutet $FF aber –1, damit ist der Inhalt des Akkumulators größer als der Inhalt der Speicherzelle. Für das N Bit gilt also A > M. Es wird ebenfalls Null gesetzt (N = 0).

Ein besonderer Vergleichsbefehl ist der BIT-Befehl.

BIT	(Bit Test)	Der Inhalt einer Speicherzelle wird mit dem Inhalt des Akkumulators durch die UND-Funktion verknüpft, ohne daß dabei der Inhalt verloren geht. Das ZERO-Bit wird entsprechend dem Ergebnis gesetzt. Bit 6 des Ergebnisses wird in das Overflow (V) Bit, Bit 7 in das Negativ (N) Bit übernommen.

3.5 Verzweigungsbefehle und Sprungbefehle

Durch einen Sprungbefehl wird ein Programm immer an einer anderen Stelle weitergeführt. Bei Verzweigungsbefehlen wird ein Programm nur dann an einer anderen Stelle weitergeführt, wenn die entsprechende Bedingung im Statusregister erfüllt ist. Der Operand nach dem Operationscode wird als Zahl im 2-er Complement aufgefasst, sodaß Verzweigungen nur um – 128 bis + 127 Zellen ausgeführt werden können.

BCC	(Branch on Carry Clear)	Verzweigung, wenn C = 0
BCS	(Branch on Carry Set)	Verzweigung, wenn C = 1
BEQ	(Branch on EQual)	Verzweigung, wenn Z = 1
BNE	(Branch Not Equal)	Verzweigung, wenn Z = 0
BMI	(Branch on Minus)	Verzweigung, wenn N = 1
BPL	(Branch on PLus)	Verzweigung, wenn N = 0

BVC (Branch on OVerflow Clear) Verzweigung, wenn V = 0

BVS (Branch on OVerflow Set) (Verzweigung, wenn V = 1

JMP (Jump) Springe unbedingt.

JSR (Jump Subroutine) Sprung in Unterprogramm.

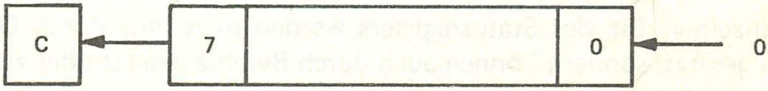
RTS (Return from Subroutine) Rücksprung aus Unterprogramm.

RTI (Return from Interrupt) Rücksprung aus Interruptprogramm.

3.6 Schiebebefehle

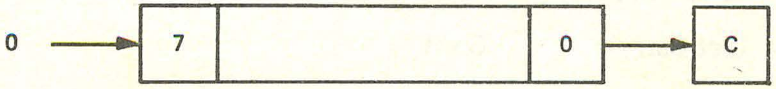
Mit diesen Befehlen wird der Inhalt einer Speicherzelle oder der Inhalt des Akkumulators um eine Stelle nach rechts oder links verschoben.

ASL (Arithmetic Shift Left) Akkumulator- oder Speicherinhalt eine Stelle links schieben Bit 7 wird in das CARRY-Bit übergeben, Bit 0 wird 0 (Abbildung 3.2).



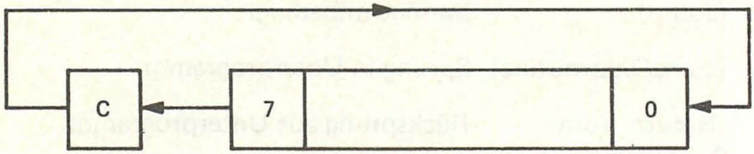
3.2 Der Befehl ASL

LSR (Logical Shift Right) Akkumulator- oder Speicherinhalt eine Stelle nach rechts schieben. Bit 0 wird in das CARRY-Bit übergeben, Bit 7 wird 0 (Abbildung 3.3).



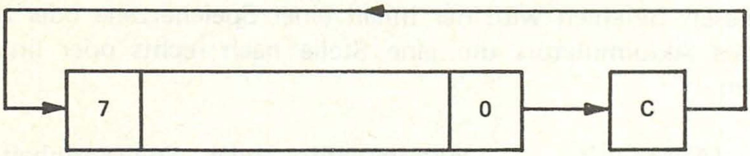
3.3 Der Befehl LSR

ROL (Rotate Left) Akkumulator- oder Speicherinhalt wird eine Stelle nach links geschoben. Dabei übernimmt Bit 0 das CARRY-Bit, dieses aber übernimmt den Inhalt von Bit 7 (Abbildung 3.4).



3.4 Der Befehl ROL

ROR (Rotate Right) Akkumulator- oder Speicherinhalt wird eine Stelle nach rechts geschoben. Das CARRY-Bit wird nach Bit 7 und Bit 0 ins CARRY-Bit geschrieben (Abbildung 3.5).



3.5 Der Befehl ROR

3.7 Statusregister-Befehle

Die einzelnen Bit des Statusregisters werden nicht nur durch Operationen gesetzt, sondern können auch durch Befehle gesetzt oder zurückgesetzt werden.

CLC (Clear Carry) C = 0

CLD (Clear Decimal) D = 0

CLI (Clear Interrupt) I = 0

CLV (Clear Overflow) V = 0

SEC (Set Carry) C = 1

SED (Set Decimal) D = 1

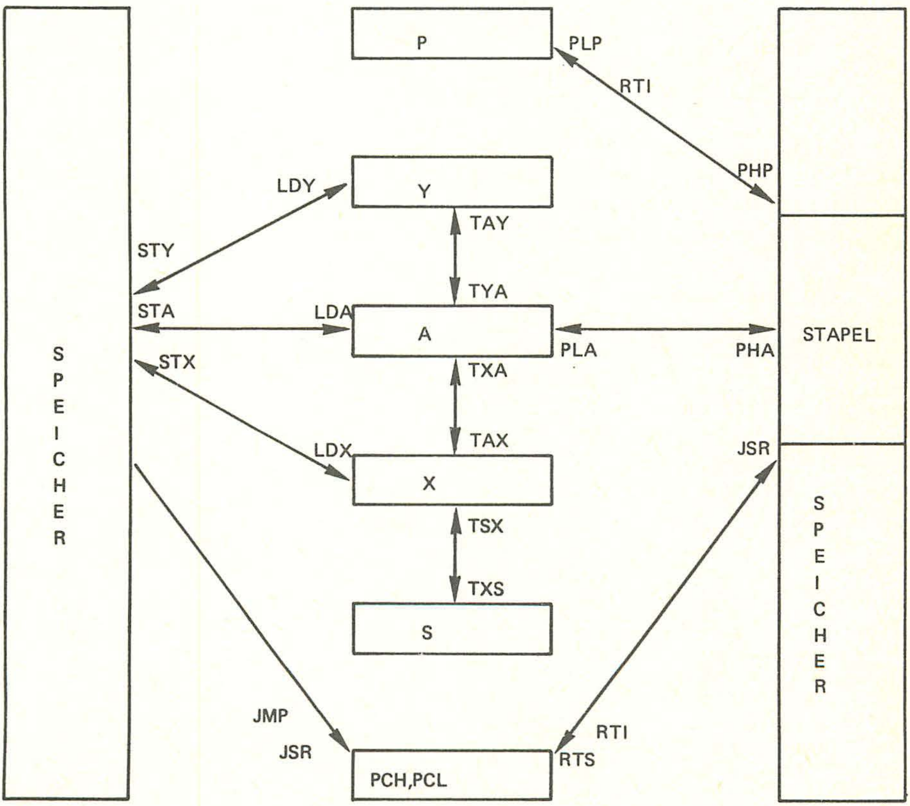
SEI (Set Interrupt) I = 1

3.8 Verschiedene Befehle

Die letzte Befehlsgruppe enthält Befehle, die verschiedene Eigenschaften haben.

NOP (No Operation) Leerbefehl

BRK (Break) Software Interrupt



3.6 Transportbefehle

In der Abbildung 3.6 sind alle Transportbefehle zwischen den Speicher

und den Registern zusammengestellt.

Adressierungs-arten

4. Die Adressierungsarten

Im letzten Kapitel haben wir die verschiedenen Befehlsarten des 6502 kennengelernt. Diese Befehle müssen nun noch durch Angabe der Adresse erweitert werden. Es muß also der Speicherplatz angegeben werden, auf welchen die Operation angewendet werden soll. Es gibt allerdings eine Befehlsgruppe, die keine Adressierung benötigt. Dies sind 1 Byte Befehle, wie TAX oder TYA. Hier ist die Adresse schon im Operationscode enthalten. TAX bedeutet, daß der Inhalt des Akkumulators in das X-Register übertragen werden soll.

Bei einer zweiten Gruppe von Befehlen genügt eine Adressenangabe. Das sind z. B. Befehle, die sich auf einen besonderen Speicherbereich beziehen, nämlich auf die SEITE NULL (ZERO PAGE) des Speichers. Darunter versteht man die ersten 256 Speicherplätze mit den Adressen $00 bis $FF. Nach diesen 2 Byte Befehlen gibt es noch Befehle, die 3 Byte belegen. Bei diesen wird in den, auf den Operationscode folgenden Byte die vollständige Adresse abgelegt. Hierbei gilt bei 6502 Systemen folgende Konvention: Bei der Adressangabe wird zuerst der niederwertige Adressteil, dann der höherwertige Adressteil gespeichert (siehe Abbildung 4.1).

OP			1 Byte Befehl, wie TAX, TAY, PLP
OP	AD		2 Byte Befehl, wie STA in ZEROPAGE
OP	ADL	ADH	3 Byte Befehl, wie LDX mit Zeile $AFCD

4.1 Befehlsaufbau beim 6502

Im Gegensatz zu anderen Prozessortypen weist der 6502 13 verschiedene Adressierungsmöglichkeiten auf, die nun im folgenden besprochen werden.

4.1 Die unmittelbare Adressierung

Diese Adressierungsart belegt 2 Byte. Dabei wird im 2. Byte nicht eine Adresse sondern ein Datenbyte gespeichert.

Beispiel: LDX # $C9
In Worten ausgedrückt, bedeutet dies: Lade das X-Register mit dem Wert $C9. Die meisten Assembler benutzen zur Kennzeichnung der unmittelbaren Adressierung das # (Number) Zeichen.

4.2 Die absolute Adressierung

Bei dieser Form der Adressierung folgt auf das Byte mit dem Operationscode die vollständige Adresse. Dabei wird im 2. Byte der niederwertige Adressteil, im 3. Byte der höherwertige Adressteil gespeichert.

Beispiel: JMP $00 $30 , Springe zur Adresse $3000.

4.3 Die absolute Adressierung der Seite 0 (Zero Page)

Für die ersten 256 Zellen eines Speichers mit den Adressen $00 bis $FF genügt nur eine Adressenangabe.

Beispiel: STA $10 , Speichere den Akkumulatorinhalt in die Zelle $10.
Diese Adressierung belegt nur 2 Byte, und die Befehlszykluszeit ist kürzer als bei der normalen absoluten Adressierung. Deshalb wird man Speicherzellen, auf die man häufig zugreift, in diese Seite 0 legen.

4.4 Die indizierte Adressierung

Die dem Operationscode folgende Adresse wird durch den Inhalt des X- oder des Y-Registers verändert. Und zwar wird der Inhalt des Indexregisters zum niederwertigen Adressbyte hinzugezählt. Dies ergibt dann die richtige Adresse.

Beispiel: LDA $300, X. Lade den Akkumulator mit der Zelle die sich aus der Summe von $300 und X-Registerinhalt ergibt. Mit X = 1 wird also der Inhalt der Zelle $301 mit X = $10 der Inhalt der Zelle $310 in den Akkumulator übernommen.
Bei der Bildung der Summe von Adressbyte und Inhalt des Indexregisters wird ein eventueller Übertrag in das 3. Byte übernommen.

Beispiel: STA $210, Y . Mit Y = $FF wird der Akkumulatorinhalt in die Zelle $30F gespeichert. Die indizierte Adressierung ist auch mit einer Adresse der Seite 0 möglich. Tritt hier aber bei der Bildung der Summe von Adresse und Indexregisterinhalt ein Übertrag auf, so wird dieser nicht berücksichtigt.

Beispiel: STA $10, X . Mit X = $FF wird der Akkumulatorinhalt in die Zelle $0F gespeichert.

4.5 Die indirekte Adressierung

Bei der indirekten Adressierung ist der dem Operationscode folgende Adressteil nicht die Zieladresse, sondern der Inhalt der durch die Adresse angegebenen Zelle ist die effektive Adresse.
Bei den 6502 System ist diese Art der Adressierung nur beim Sprungbefehl implementiert.

Beispiel: JMP ($300) , Springe auf die Adresse, die als Inhalt in den Zellen $300 und $301 gespeichert ist. Die Angabe der Adressen, eingeschlossen in runden Klammern, bedeutet, daß nicht die Adresse $300 sondern der Inhalt dieser Adresse das Sprungziel ist.

Ein schönes Beispiel für die Verwendung des indirekten Sprunges findet man in der Monitor-Routine des APPLE II Computers, die ein Zeichen auf den Bildschirm ausgibt.

```
        .
        .
        .
$FDED   JMP (CSWL)
$FDFO   CMP #$A0
        .
        .
        .
```

In Zelle $FDED ist ein indirekter Sprung über die mit CSWL bezeichnete Zelle programmiert. Schaut man in dieser Zelle nach, welchen Inhalt sie hat, so findet man dort die Adresse $FDF0, also die Adresse des folgenden Befehles. Diese Art Der Programmierung hat folgenden Zweck. Jedesmal, wenn ein Zeichen auf den Bildschirm ausgegeben werden soll, wird diese Befehlsfolge durchlaufen. Möchte man das Zeichen aber auch noch auf ein anderes Gerät ausgeben, z. B. auf einen Drucker, so wird der Inhalt der Zelle CSWL durch die Adresse der Druckerausgabe ersetzt und am Ende dieses Programmteils auf die Zelle $FDF0 gesprungen.

Eine Kombination von indirekter und indizierter Adressierung bildet die indiziert-indirekte und die indirekt-indizierte Adressierung. In beiden Fällen ist die Basisadresse immer eine Adresse aus der Seite 0.

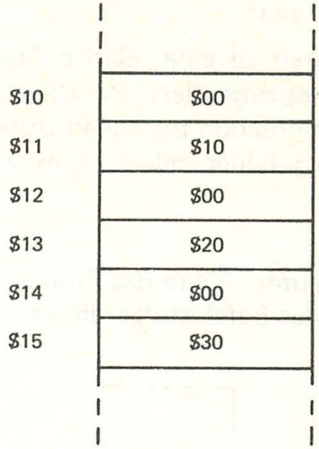
4.6 Die indiziert-indirekte Adressierung

Diese Art der Adressierung macht man sich am besten an einem Beispiel klar:

LDA ($10, X) bedeutet: Zuerst wird zum Adressbyte $10 der Inhalt des Indexregisters addiert. Dies ergibt eine neue Adresse, deren Inhalt die eigentliche Zieladresse darstellt. Abbildung 4.2 soll dies verdeutlichen.

In den Zellen $10 und $11 ist die Adresse $1000, in den Zellen $12 und $13 die Adresse $2000 und in den Zellen $14 und $15 die Adresse $3000 gespeichert (Niederwertiges Adressbyte immer zuerst). Der Befehl LDA ($10, X) bewirkt nun, daß mit X = 0 der Inhalt der Speicherzelle $1000, mit X = 2 der Inhalt der Zelle $2000 und mit X = 4 der Inhalt der Zelle $3000 in den Akkumulator übernommen wird.

Für diese Art der Indizierung kann nur das X-Register verwendet werden. Tritt bei der Summenbildung Adressbyte und Inhalt des Indexregisters ein Übertrag auf, so wird dieser nicht berücksichtigt.

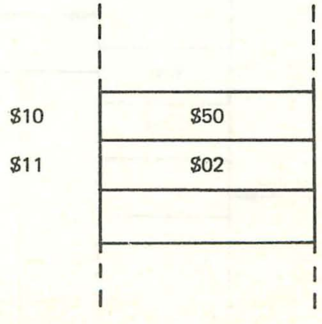


4.2 Indiziert-indirekte Adressierung

4.7 Die indirekt-indizierte Adressierung

Für die Indizierung wird hier das Y-Register verwendet. Der Inhalt des Indexregisters wird aber hier nicht zum Adressbyte, sondern zum Inhalt des Adressbytes addiert. Entsteht bei dieser Addition ein Übertrag, so wird dieser in der Speicherzelle Adressbyte +1 berücksichtigt.

Beispiel: STA ($10), Y . In der Speicherzelle $10 ist $50 und in der Speicherzelle $11 der Wert $02 (Abbildung 4.3) gespeichert. Mit Y = 0 wird nun der Akkumulatorinhalt in die Zelle $250, mit Y = $FF der Akkumulatorinhalt in die Zelle $34F gespeichert.

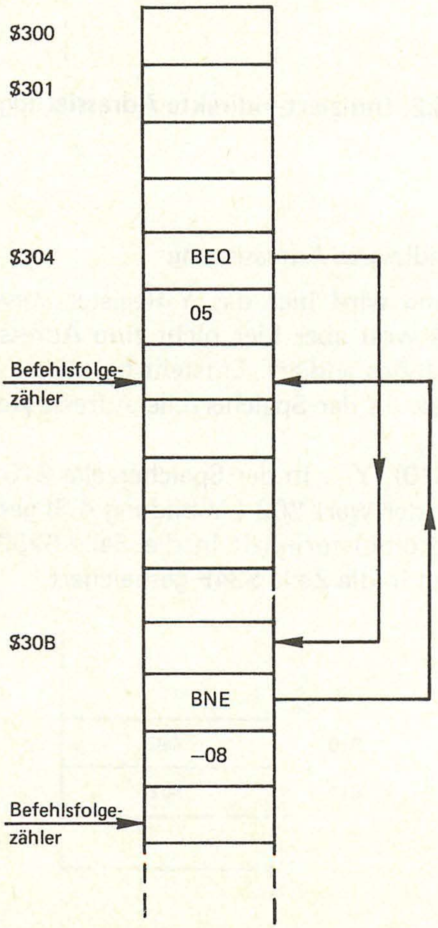


4.3 Indirekt-indizierte Adressierung

4.8 Die relative Adressierung

Bei den 6502 Prozessoren ist eine relative Adressierung nur bei den Verzweigungsbefehlen implementiert. Bei dieser Art der Adressierung wird im auf den Operationscode folgenden Byte eine Zahl gespeichert, mit der der Befehlsfolgezähler relativ zum augenblicklichen Stand verändert wird.

Beispiel: BEQ 05 bedeutet: Führe das Programm 5 Zellen nach dem augenblicklichen Stand des Befehlsfolgezählers weiter (Abbildung 4.4).



4.4 Relative Adressierung

In dem angegebenen Beispiel steht der Befehlsfolgezähler bei der Entschlüsselung des Befehls BEQ 05 schon in Zelle $307. Falls die Bedingung für die Verzweigung zutrifft wird das Programm in Zelle $30B weitergeführt. Der Rückwärtssprung BNE –08 führt auf die Zelle $306, denn der Befehlsfolgezähler zeigt schon auf die Zelle $30E. Der Zahlenwert im Adressbyte wird also als vorzeichenbehaftete Zahl interpretiert. Damit sind Programmverzweigungen nur im Bereich – 128 bis + 127 Byte möglich.

Bei kurzen, von hand assemblierten Programmen ist die Berechnung der Sprungziele immer wieder eine häufige Fehlerquelle.
Zur Vereinfachung dieser Berechnung sind in Abbildung 4.5 und 4.6 zwei Tabellen zur Bestimmung der Sprungdistanz angegeben. Man zählt vom augenblicklichen Stand, die zu überspringenden Zellen ab, und sucht sich den entsprechenden Hexadezimalwert aus der Tabelle heraus.

LSD / MSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

4.5 Vorwärts-Verzweigungen

LSD / MSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	128	127	126	125	124	123	122	121	120	119	118	117	116	115	114	113
9	112	111	110	109	108	107	106	105	104	103	102	101	100	99	98	97
A	96	95	94	93	92	91	90	89	88	87	86	85	84	83	82	81
B	80	79	78	77	76	75	74	73	72	71	70	69	68	67	66	65
C	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49
D	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
E	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
F	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

4.6 Rückwärtzs-Verzweigungen

4.9 Die implizierte Adressierung

Bei einigen Befehlen ist die Angabe einer Adresse nicht notwendig, da diese schon im Operationscode enthalten ist.

Beispiel: DEY Decrement Y-Register.

4.10 Die Akkumulatorbezogene Adressierung

Befehle, die direkt den Akkumulator betreffen benötigen ebenfalls keine weiteren Adressenangaben. Beim 6502 Prozessor sind dies nur Schiebebefehle.

Beispiel: LSR Schiebe Akkumulatorinhalt eine Stelle nach rechts.

Bis jetzt haben wir für die Befehle die memnotechnischen (leicht merkbaren) Ausdrücke verwendet. Für den Prozessor sind dies aber Bitmuster. In Abbildung 4.7 sind alle Befehle zusammengestellt, wobei die Bitmuster als Hexadezimalzahlen angegeben sind.

Die Rückumwandlung von der Darstellung durch Hexadezimalzahlen in die memnotechnischen Bezeichnungen erfolgt durch Abbildung 4.8.

Befehle	symb. Code	Wirkung	ADRESSIERUNGSARTEN													N	Z	C	I	D	V
			UNM	ABS	ABS,X	ABS,Y	S0	S0,X	S0,Y	(IND,X)	(IND),Y	REL	IND	ACCU	IMPL						
Transport	LDA	M → A	A9	AD	BD	B9	A5	B5		A1	B1					X	X	–	–	–	–
	LDX	M → X	A2	AE		BE	A6		B6							X	X	–	–	–	–
	LDY	M → Y	A0	AC	BC		A4	B4								X	X	–	–	–	–
	STA	A → M		8D	9D	99	85	95		81	91					–	–	–	–	–	–
	STX	X → M		8E			86		96							–	–	–	–	–	–
	STY	Y → M		8C			84	94								–	–	–	–	–	–
	TAX	A → X													AA	X	X	–	–	–	–
	TAY	A → Y													A8	X	X	–	–	–	–
	TXA	X → A													8A	X	X	–	–	–	–
	TYA	Y → A													98	X	X	–	–	–	–
	TXS	X → S													9A	–	–	–	–	–	–
	TSX	S → X													BA	X	X	–	–	–	–
	PLA	S+1 → S, Ms → A													68	X	X	–	–	–	–
	PHA	A → Ms, S–1 → S													48	–	–	–	–	–	–
	PLP	S+1 → S, Ms → P													28						
	PHP	P → Ms, S–1 → S													08	–	–	–	–	–	–
Arithmetische	ADC	A+M+C → A	69	6D	7D	79	65	75		61	71					X	X	X	–	–	X
	SBC	A–M–$\overline{C}$ → A	E9	ED	FD	F9	E5	F5		E1	F1					X	X	X	–	–	X
	INC	M+1 → M		EE	FE		E6	F6								X	X	–	–	–	–
	DEC	M–1 → M		CE	DE		C6	D6								X	X	–	–	–	–
	INX	X+1 → X													E8	X	X	–	–	–	–
	DEX	X–1 → X													CA	X	X	–	–	–	–
	INY	Y+1 → Y													C8	X	X	–	–	–	–
	DEY	Y–1 → Y													88	X	X	–	–	–	–
Logische	AND	A ∧ M → A	29	2D	3D	39	25	35		21	31					X	X	–	–	–	–
	ORA	A ∨ M → A	09	0D	1D	19	05	15		01	11					X	X	–	–	–	–
	EOR	A ⊻ M → A	49	4D	5D	59	45	55		41	51					X	X	–	–	–	–
Vergleichs-	CMP	A–M	C9	CD	DD	D9	C5	D5		C1	D1					X	X	X	–	–	–
	CPX	X–M	E0	EC			E4									X	X	X	–	–	–
	CPY	Y–M	C0	CC			C4									X	X	X	–	–	–
	BIT	A ∧ M		2C			24									7	X	–	–	–	6
Verzweigungs-	BCC	BRANCH ON C=0									90					–	–	–	–	–	–
	BCS	BRANCH ON C=1									B0					–	–	–	–	–	–
	BEQ	BRANCH ON Z=1									F0					–	–	–	–	–	–
	BNE	BRANCH ON Z=0									D0					–	–	–	–	–	–
	BMI	BRANCH ON N=1									30					–	–	–	–	–	–
	BPL	BRANCH ON N=0									10					–	–	–	–	–	–
	BVC	BRANCH ON V=0									50					–	–	–	–	–	–
	BVS	BRANCH ON V=1									70					–	–	–	–	–	–
	JMP		4C										6C			–	–	–	–	–	–
	JSR		20													–	–	–	–	–	–
Schiebe-	ASL		0E	1E		06	16							0A		X	X	X	–	–	–
	LSR		4E	5E		46	56							4A		X	X	X	–	–	–
	ROL		2E	3E		26	36							2A		X	X	X	–	–	–
	ROR		6E	7E		66	76							6A		X	X	X	–	–	–
Status-Register-	CLC	C=0													18	–	–	0	–	–	–
	CLD	D=0													D8	–	–	–	–	0	–
	CLI	I=0													58	–	–	–	0	–	–
	CLV	V=0													B8	–	–	–	–	–	0
	SEC	C=1													38	–	–	1	–	–	–
	SED	D=1													F8	–	–	–	–	1	–
	SEI	I=1													78	–	–	–	1	–	–
Versch.	NOP	NO OPER													EA	–	–	–	–	–	–
	RTS	RETURN F. SUB													60	–	–	–	–	–	–
	RTI	RETURN F. INT													40						
	BRK	BREAK													00	–	–	–	1	–	–

4.7 Befehlsliste der 6502 CPU

MSD \ LSD	0	1	2	3	4	5	6	7
0	BRK	ORA–IND, X				ORA–Z PAGE	ASL–Z PAGE	
1	BPL	ORA–IND, Y				ORA–Z PAGE, X	ASL–Z PAGE	
2	JSR	AND–IND, X			BIT–Z Page	AND–Z PAGE,	ROL–Z PAGE	
3	BMI	AND–IND, Y				AND–Z PAGE, X	ROL–Z PAGE, X	
4	RTI	EOR–IND, X				EOR–Z PAGE	LSR–Z PAGE	
5	BVC	EOR–IND, Y				EOR–Z PAGE, X	LSR–Z PAGE, X	
6	RTS	ADC–IND, X				ADC–Z PAGE	ROR–Z PAGE	
7	BVS	ADC–IND, Y				ADC–Z PAGE, X		
8		STA–IND, X			STY–Z Page	STA–Z PAGE	STX–Z PAGE	
9	BCC	STA–IND, Y			STY–Z Page, X	STA–Z PAGE, X	STX–Z PAGE, Y	
A	LDY–IMM	LDA–IND, X	LDX–IMM		LDY–Z Page	LDA–Z PAGE	LDX–Z PAGE	
B	BCS	LDA–IND, Y			LDY–Z Page, X	LDA–Z PAGE, X	LDX–Z PAGE, Y	
C	CPY–IMM	CMP–IND, X			CPY--Z Page	CMP–Z PAGE	DEC–Z PAGE	
D	BNE	CMP–IND, Y				CMP–Z PAGE, X	DEC–Z PAGE, X	
E	CPX–IMM	SBC–IND, X			CPX–Z Page	SBC–Z PAGE	INC–Z PAGE	
F	BEQ	SBC–IND, Y				SBC–Z PAGE, X	INC–Z PAGE, X	

MSD \ LSD	8	9	A	B	C	D	E	F
0	PHP	ORA–IMM	ASL–A			ORA–ABS	ASL–ABS	
1	CLC	ORA–ABS, Y				ORA–ABS, X	ASL–ABS, X	
2	PSP	AND–IMM	ROL–A		BIT–ABS	AND–ABS	ROL–ABS	
3	SEC	AND–ABS, Y				AND–ABS, X	ROL–ABS, X	
4	PHA	EPR–IMM	LSR–A		JUMP–ABS	EOR–ABS	LSR–ABS	
5	CLI	EOR–ABS, Y				EOR–ABS, X	LSR–ABS, X	
6	PLA	ADC–IMM	ROR–A		JMP–IND	ADC–ABS	ROR–ABS	
7	SEI	ADC–ABS, Y				ADC–ABS, X		
8	DEY		TXA		STY–ABS	STA–ABS	STX–ABS	
9	TYA	STA–ABS, Y	TXS			STA–ABS, X		
A	TAY	LDA–IMM	TAX		LDY–ABS	LDA–ABS	LDX–ABS	
B	CLV	LDA–ABS, Y	TSX		LDY–ABS, X	LDA–ABS, X	LDX–ABS, Y	
C	INY	CMP–IMM	DEX		CPY–ABS	CMP–ABS	DEC–ABS	
D	CLD	CMP–ABS, Y				CMP–ABS, X	DEC–ABS, X	
E	INX	SBC–IMM	NOP		CPX–ABS	SBC–ABS	INC–ABS	
F	SED	SBC–ABS, Y				SBC–ABS, X	INC–ABS, X	

4.8 Bitmusterverteilung der 6502 CPU

Programm-
entwicklung

5. Programmentwicklung

Programmieren bedeutet das Umsetzen einer Aufgabe in die Sprache des Computers. Ist diese Sprache eine höhere Programmiersprache, so ist dieses Umsetzen relativ einfach. Ein Umsetzen in den Maschinencode bedeutet aber eine viel genauere Formulierung der Aufgabe und ein Zerlegen in einzelne Schritte, die den Befehlen des Computers entsprechen.

Betrachten wir hierzu folgende Aufgabe:

> Addiere die Zahlen 3 und 5 und gebe das Ergebnis auf den Bildschirm aus.

Die Lösung in BASIC ist einfach:

```
10 A = 3
20 B = 5
30 PRINT A + B
40 END
```

Die Lösung dieser Aufgabe durch Maschinensprache erfordert aber vor der Programmierung noch weitere Angaben:

a) Werden diese Zahlen durch 8-bit, 16-bit oder durch ein Gleitkommaformat dargestellt?

b) Sind es Zahlen mit oder ohne Vorzeichen?

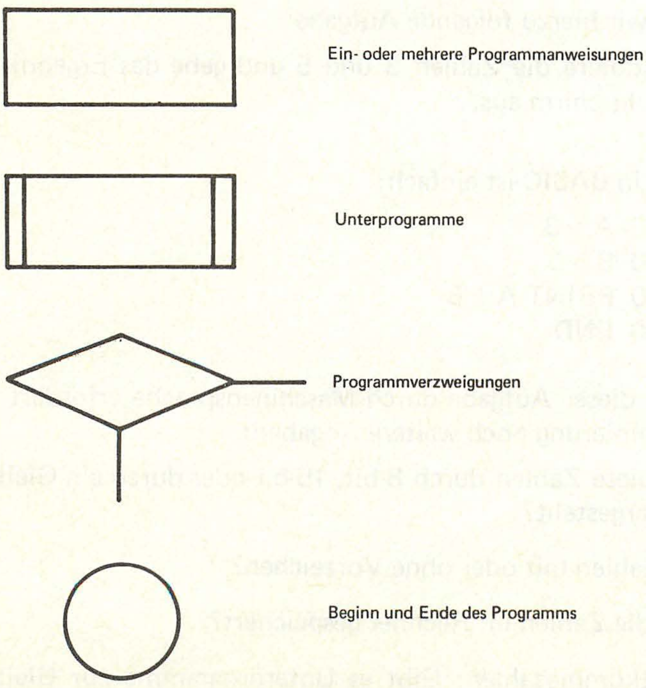
c) Wo sind die Zahlen im Rechner gespeichert?

d) Bei Gleitkommazahlen: Gibt es Unterprogramme zur Gleitkommaaddition und zur Ausgabe einer Gleitkommazahl auf den Bildschirm?

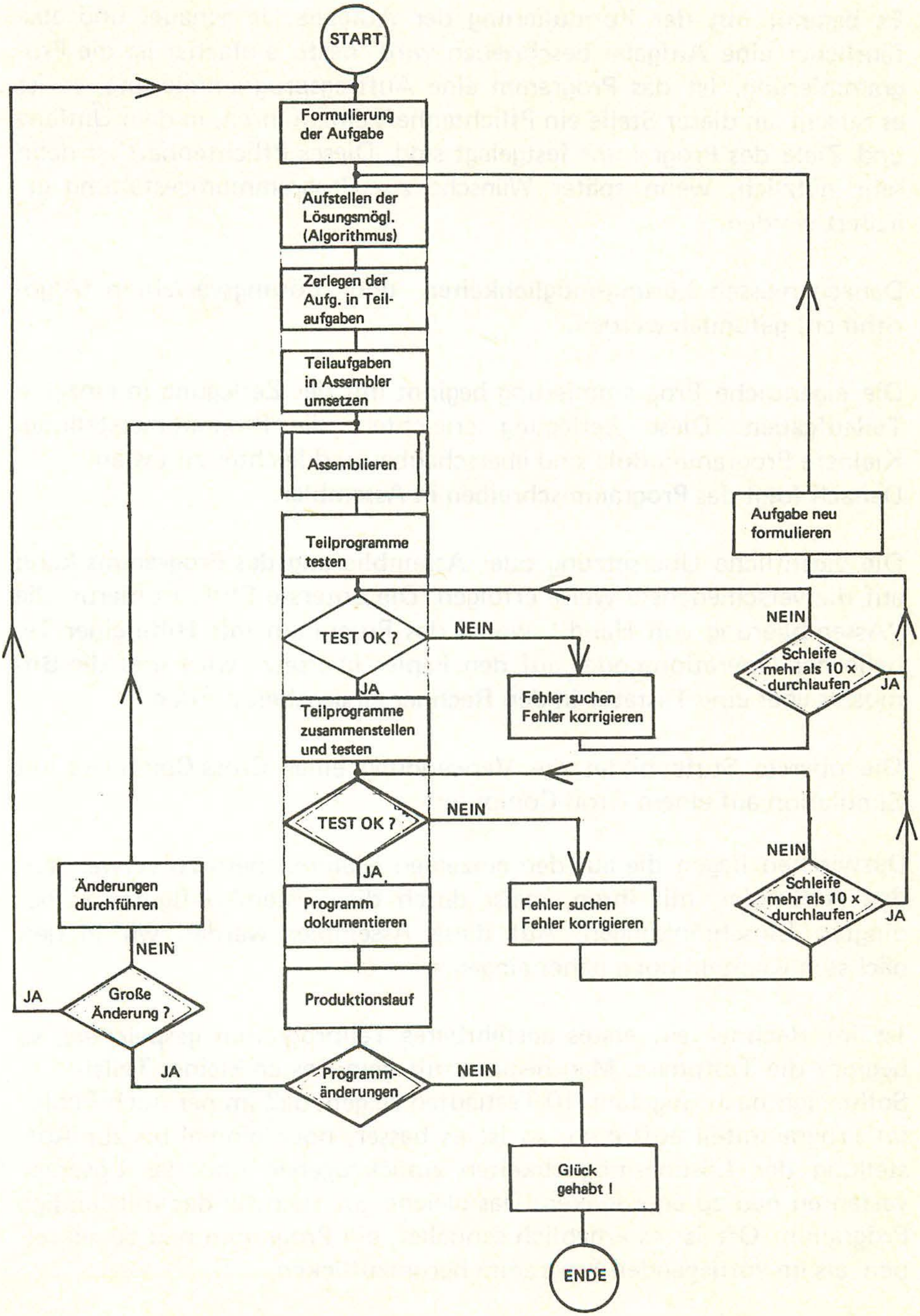
c) Bei 8-bit oder 16-bit Zahlen: Gibt es ein Unterprogramm zur Ausgabe dieser Zahl auf den Bildschirm?

Erst wenn diese Fragen geklärt sind, kann das Programm geschrieben werden.

Umfangreiche Programme wird man zuerst in Teilaufgaben zerlegen und auf den Papier durch ein Flußdiagramm oder ein Struktogramm darstellen. Die Elemente eines Flußdiagramms sind in Abbildung 5.1 zusammengestellt. In einem Rechteck werden ein oder mehrere Anweisungen angeführt. Bilden diese Anweisungen ein vollständiges Unterprogramm, so werden die Seiten durch zwei Striche markiert. Eine Abfrage wird durch eine Raute dargestellt. Die Ausgänge werden bezeichnet (JA/NEIN oder Bedingung erfüllt, Bedingung nicht erfüllt). Ein Kreis bildet Ein- oder Ausgang des Programms oder eines Unterprogramms. Abbildung 5.2 ist das Flußdiagramm zur "Erstellung eines Programms". Diese einzelnen Schritte wollen wir uns mal näher betrachten:



5.1 Flußdiagramm-Symbole



5.2 Flußdiagramm: "Erstellung eines Programms"

Es beginnt mit der **Formulierung der Aufgabe**. Je genauer und ausführlicher eine Aufgabe beschrieben wird, desto einfacher ist die Programmierung. Ist das Programm eine Auftragsprogrammierung, so ist es ratsam, an dieser Stelle ein Pflichtenheft einzuführen, in dem Umfang und Ziele des Programms festgelegt sind. Dieses Pflichtenheft ist dann sehr nützlich, wenn später Wünsche zur Programmumgestaltung geäußert werden.

Danach müssen **Lösungsmöglichkeiten** und **Lösungsverfahren** (Algorithmen) gefunden werden.

Die eigentliche Programmierung beginnt mit der Zerlegung in einzelne **Teilaufgaben**. Diese Zerlegung erleichtert die Programmerstellung. Kleinere Programmodule sind überschaubar und leichter zu testen.
Danach folgt das **Programmschreiben in Assembler**.

Die eigentliche Übersetzung oder **Assemblierung des Programms** kann auf die verschiedenste Weise erfolgen. Die unterste Stufe ist hierbei die "Assemblierung von Hand", wobei das Programm mit Hilfe einer Tabelle der Operationscodes, auf den Papier übersetzt wird und die Bitmuster über eine Tastatur in den Rechner eingegeben werden.

Die oberste Stufe bildet die Verwendung eines Cross-Compilers mit Simulation auf einem Groß-Computer.

Dazwischen liegen die auf den einzelnen Kleinrechnern zu verwendenden Assembler, mit ihren, meist durch die Systemkonfiguration bedingten Einschränkungen. Auf diese Assembler werden wir in den nächsten Kapiteln noch näher eingehen.

Ist im Rechner ein erstes ausführbares Teilprogramm gespeichert, so beginnt die Testphase. Man beginnt mit dem Testen kleiner Teilstücke. Sollte sich nach ungefähr 10 Testläufen zeigen, daß immer noch Fehler im Programmteil auftreten, so ist es besser, noch einmal bis zur Aufstellung der Lösungsmöglichkeiten zurückzugehen und das Lösungsverfahren neu zu überdenken. Das gleiche gilt auch für das vollständige Programm. Oft ist es erheblich schneller, ein Programm neu zu schreiben, als im vorliegenden Programm herumzuflicken.

Sind alle Fehler beseitigt, das Programm läuft, so beginnt ein weiterer

wichtiger Punkt: Die **Programmdokumentation**. Dieser Punkt wird in vielen Fällen übergangen (Man weiß ja, was man programmiert hat !). Diese Fehleinschätzung rächt sich später oft bitter.

Eine Programmdokumentation solte folgende Punkte enthalten:

Eine Kurzbeschreibung der Aufgabe (Programmspezifikation),
eine Beschreibung der verwendeten Algorithmen,
eine Beschreibung der Unterprogramme,
einen Speicherbelegungsplan, mit Angabe der verwendeten Speicherzellen und ihrer Bedeutung und
die verwendeten Flußdiagramme.

Die letzte Abfrage bezieht sich dann auf Programmänderungen. Werden keine Änderungen gewünscht, hat man Glück gehabt, und ist mit der Programmierung fertig.

In den meisten Fällen werden aber Programmänderungen gewünscht. Kleine Änderungen werden in den Teilprogrammen durchgeführt und man beginnt neu mit der Assemblierung. Bei großen Änderungswünschen beginnt man am bestem mit einer Neuprogrammierung. Bei solch einen Fall erweist sich die Vorlage eines Pflichtenheftes als äußerst nützlich.

Zwei alte Programmierweisheiten werden aber immer bleiben:

1. Jede kleine Programmänderung hat mindestens 10 große Änderungen zur Folge.

2. Bei der ersten Vorführung des Programms wird zum erstenmal jene Programmschleife durchlaufen, die noch einen Fehler enthält.

5.1 Aufbau eines Assemblerprogramms

Das Niederschreiben eines Programms erfolgt am besten in der Schreibweise, wie sie auch für die Eingabe in die Assembler verwendet wird. Leider hat sich hierfür keine einheitliche Formulierung gebildet. Es gibt Unterschiede, auf die noch eingegangen werden muß. Im allgemeinen wird aber eine Eingabezeile in einen Assembler folgendermaßen auf-

geteilt:

MARKENFELD	ASS. BEFEHL	ADRESSIERUNG	KOMMENTAR

Für Programme in diesem Buch, die sich nicht auf einen bestimmten Assembler beziehen, soll folgende Vereinbarung gelten: Das Markenfeld kann bis 6 Zeichen lang sein, das erste Zeichen muß ein Buchstabe sein. Es folgt die Angabe des memnonischen Codes mit 3 Zeichen und anschließend die Art der Adressierung. Das Kommentarfeld wird vom Adressfeld durch einen Strichpunkt (;) getrennt. Dies ist eine Konvention, die bei vielen Assemblern anzutreffen ist.

5.2 Ein erstes Beispiel und "Assemblierung von Hand"

Als erstes Bespiel wollen wir das im vorigen Kapitel angeführte Beispiel von der Addition zweier Zahlen (3 + 5) in Maschinencode programmieren.

Dazu folgende Angaben:

> 3 und 5 sind 16-bit Zahlen ohne Vorzeichen. Die Zahl 3 ist in Zelle \$10 und \$11, die Zahl 5 ist in Zelle \$12 und \$13 gespeichert. Diese Speicherung erfolgt immer in der Reihenfolge LSB, MSB.

Zur Ausgabe auf den Bildschirm wird ein Unterprogramm PRTBYT verwendet, das den Inhalt eines Speicherplatzes als 2 Hexadezimalzahlen ausgibt. Dieses Unterprogramm, sowie auch die Rücksprungadresse MONITO sind bei jedem 6502 Computer verschieden.

Assemblerprogramm:

```
LDA     #$03   ;  3 → Akkumulator
STA      $10   ;  Akkumulator → Zelle $10
LDA     #$05   ;  5 → Akkumulator
STA      $12   ;  Akkumulator → Zelle $12
LDA     #$00   ;  0 → Akkumulator
STA      $11   ;  Akkumulator → $11
STA      $13   ;  Akkumulator → $13
CLC            ;  Löschen des Carry-Bit
LDA      $10   ;  Inhalt der Zelle $10 → Akkumulator
ADC      $12   ;  Addiere dazu den Inhalt von Zelle $12
STA      $10   ;  Ergebnis nach Zelle $10 speichern
LDA      $11   ;  Lade Akkumulator mit Inhalt von Zelle $11
ADC      $13   ;  Addiere dazu den Inhalt von Zelle $13
STA      $11   ;  Ergebnis nach Zelle $11
JSR   PRTBYT   ;  und Ausgabe auf Bildschirm (MSB zuerst)
LDA      $10   ;  Inhalt von $10 → Akkumulator
JSR   PRTBYT   ;  Ausgabe auf Bildschirm
JMP   MONITO   ;  Sprung in den Monitor .
```

5.4 Addition der Zahlen 3 und 5

Dieses Programm soll nun mit der Papier und Bleistift Methode in den Maschinencode übertragen werden. Dazu benötigen wir ein Blatt karriertes Papier und die Operationscode-Tabelle in Abbildung 4.7. Man kann auch ein Codierblatt (Abbildung 5.6) benützen.

Bei dieser Umsetzung ist als Kommentar nicht wie in Abbildung 5.4 die Wirkung des Befehls, sondern die Adressierungsart angegeben. Vorher muß noch festgelegt werden, an welchem Speicherplatz das Programm beginnen soll. Die Startadresse, also die Adresse des ersten gültigen Operationscodes, soll die Zelle $800 sein. Eine Zeile des assemblierten Programms hat dann folgendes Aussehen:

Speicher-zelle	Code	Marke	Ass. Befehl	Adressierung	Kommentar

Assembliertes Programm:

```
$800  A9 03       LDA  #$03   ; unmittelbare Adressierung
$802  85 10       STA   $10   ; absolute Adressierung, Zero-P.
$804  A9 05       LDA  #$05
$806  85 12       STA   $12
$808  A9 00       LDA  #$00
$80A  85 11       STA   $11
$80C  85 13       STA   $13
$80E  18          CLC         ; implizierte Adressierung
$80F  A5 10       LDA   $10   ; absolute Adressierung, Zero-P.
$811  65 12       ADC   $12   ; absolute Adressierung, Zero-P.
$813  85 10       STA   $10
$815  A5 11       LDA   $11
$817  65 13       ADC   $13
$819  85 11       STA   $11
$81B  20 DA FD 1) JSR  PRNBYT ; Unterprogrammsprung
$80E  A5 10       LDA   $10
$820  20 DA FD    JSR  PRNBYT
$823  4C 59 FF    JMP  MONITO ; Rücksprung in den Monitor
```

```
0800- A9 0A 08 10 A9 05 85 12
0808- A9 00 32 08 85 13 18 A5
0810- 10 65 12 85 10 A5 11 65
0818- 13 85 11 20 DA FD A5 10
0820- 20 DA FD 4C 59 FF
*
```

5.5 Assembliertes Programm

---

1) Anmerkung:
Die hier angegebenen Adressen beziehen sich auf den Monitor des APPLE II Computers. Im Kapitel 8, Monitor-Routinen sind diese Unterprogramm-Adressen für andere 6502-Systeme angegeben.

Dieses Programm ist recht einfach zu codieren, denn es treten keine Marken, und somit keine Verzweigungen auf. Der Maschinencode ist nun zu Papier gebracht und soll nun in den Rechner eingegeben werden. Diese Eingabe kann nur über das Betriebssystem des Rechners, dem Monitor erfolgen.

5.3 Monitor

Der rein elektronische Zusammenbau CPU (Central-Prozessing Unit), RAM (Random Access Memory) und I/O-Baustein stellt noch kein lauffähiges Mikro-Prozessorsystem dar. Dazu wird noch ein Betriebssystem benötigt, daß in einem ROM (Read Only Memory) abgelegt ist und dafür sogt, daß Programme z. B. von Kassette eingelesen oder gespeichert werden können, daß Eingaben über Tastenfeld gemacht werden oder Zeichen über den Bildschirm ausgegeben werden. Beim Einschalten des Rechners wird automatisch ein Sprung in ein Programm ausgeführt, dessen Adresse in den Zellen $FFFC und $FFFD (RESET-Vektor) gespeichert ist.

Dieses Programm ruft entweder sofort BASIC auf (APPLE, CBM, ATARI, VC20) oder wartet auf eine Eingabe (C1P, AIM 65, SYM, KIM). Die Befehle für die Eingabe von Hexadezimalzahlen, Ändern des Inhalts von Speicherzellen usw. werden für die einzelnen Rechner im Kapitel 8 beschrieben. Hier wollen wir Monitor-Befehle des APPLE II-Rechners benutzen. Der Code des Programms in Abbildung 5.5 soll in den Rechner geschrieben und ausgeführt werden. Dazu wird mit CALL-151 der Monitor aufgerufen und danach mit 800: die Anfangsadresse des Programms und die Bitmuster der Operationscode, getrennt durch einen Zwischenraum eingegeben. Die Speicherzellen von $800 bis $825 haben dann folgenden Inhalt (Abbildung 5.5). Mit 800G wird das Programm gestartet. Als Ausdruck erscheint das Ergebnis auf dem Bildschirm: 0008.

ELCOMP Software

Aufgabe		Datum	Blatt

Speicher-zelle	Code	Marke	Ass. Befehl	Adressierung	Kommentar

symb. Adressen

symb. Namen

5.6 Codierblatt

5.4 Zweites Beispiel: Eingabe von Hexadezimalzahlen

Das vorige Beispiel wird nun erweitert. Die Zahlen sollen über das Tastenfeld eingegeben werden. Die Addition zweier 16-Bit Zahlen wird als Unterprogramm geschrieben. Dabei wird ein Fehler angezeigt, wenn bei dieser Addition ein Übertrag auftritt, d. h. wenn sich das Ergebnis nicht mehr als 16-bit Zahl darstellen lässt.

Zuerst die Eingaberoutine:
Ein Unterprogramm im Monitor, GETCHR, fragt laufend das Tastenfeld ab, ob eine Taste betätigt wird. Ist dies geschehen, so wird dieses Zeichen als ASCII (American Standard Code for Information Interchange) Zeichen in den Akkumulator übernommen. In Abbildung 5.7 sind die ASCII-Zeichen zusammengestellt.

LSD \ MSB		0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
0	0000	NUL	DLE	SP	0	@	P		p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	–	=	M	]	m	}
E	1110	SO	RS	.	>	N	↑	n	~
F	1111	SI	VS	/	?	0	←	o	DEL

5.7 ASCII-Zeichen

Die Hexadezimalzahlen 0 bis 9 werden durch die Zeichen 30 bis 39 (APPLE: B0 bis B9), A bis F durch die Zeichen 41 bis 46 (APPLE: C1 bis C6) dargestellt. In dem Unterprogramm PACK wird das durch GETCHR in den Akkumulator übernommene Zeichen überprüft, ob es eine Hexadezimalzahl darstellt. Trifft dies zu, wird es in den Eingabepuffer (zwei Zellen in der Zero-Page) übergeben. Ist es keine Hexa-

dezimalzahl, wird das Unterprogramm, mit dem ASCII-Zeichen im Akkumulator, verlassen.

Bei der Formulierung dieser Aufgabe werden wir für Sprungziele (Marken) und Speicherzellen symbolische Namen verwenden. In dem Kommentar wird mit ⟨ A ⟩ der Inhalt des Akkumulators bezeichnet. Genauso wollen wir mit ⟨ M ⟩ den Inhalt einer Speicherzelle, mit ⟨ X ⟩ den Inhalt des X-Registers und mit ⟨ Y ⟩ den Inhalt des Y-Registers bezeichnen.
Die Assemblierung erfolgt wiederum mit der Papier und Bleistift Methode.

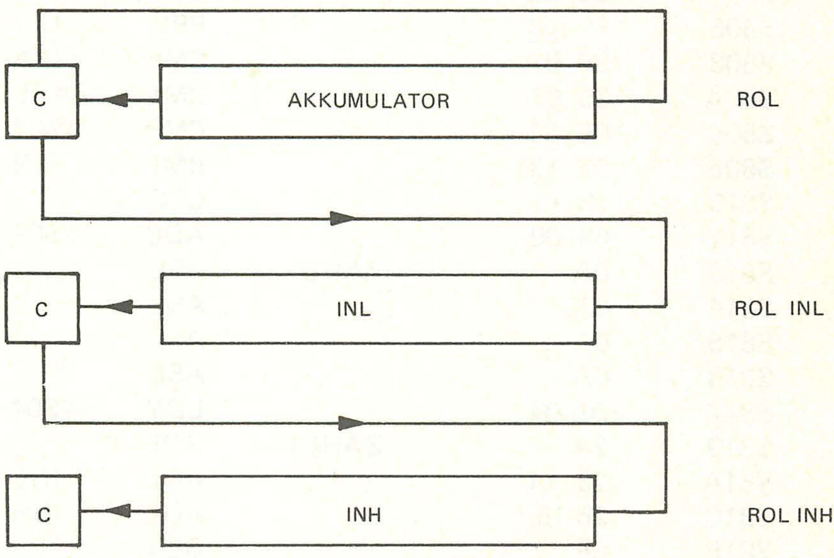
Unterprogramm PACK:

```
PACK    CMP  #$B0  ; Vergleiche ⟨ A ⟩ mit $B0
        BMI   FIN  ; wenn kleiner, keine Hex-Zahl
        CMP  #$C6  ; Vergleiche ⟨ A ⟩ mit $C6
        BPL   FIN  ; wenn größer, keine Hex-Zahl
        CMP  #$BA  ; Vergleiche ⟨ A ⟩ mit $BA
        BMI  ZAHL  ; wenn kleiner, dann Zahl von 0 – 9
        CMP  #$C1  ; Vergleiche ⟨ A ⟩ mit $C1
        BMI   FIN  ; wenn kleiner, keine Hex-Zahl
        CLC        ; sonst Hex-Zahl A bis F, damit
        ADC  #$09  ; ⟨ A ⟩ + 9 notwendig
ZAHL    ASL        ; ⟨ A ⟩ 4 mal nach links
        ASL        ; schieben. Wenn vorher ⟨ A ⟩ = $C1
        ASL        ; ist danach ⟨ A ⟩ = $10
        ASL        ;
        LDY  #$04  ; ⟨ Y ⟩ = 4
ZAHL1   ROL        ; Der ⟨ A ⟩ wird nach INL und
        ROL   INL  ; ⟨ INL ⟩ wird  nach INH geschoben
        ROL   INH  ; INL, INH Eingabepuffer
        DEY        ; ⟨ Y ⟩ = ⟨ Y ⟩ – 1
        BNE ZAHL1  ; Wenn ⟨ Y ⟩ ≠ 0; Rücksprung nach
        LDA  #$00  ; Zahl1; 0 → A
FIN     RT$        ; Rückkehr ins Hauptprogramm
```

5.8 Unterprogramm PACK

Hier bedarf die Befehlsfolge ROL
ROL INL
ROL INH noch
eine Erläuterung. Der Befehl ROL wird in Abbildung 3.4 erläutert.

Der Inhalt des Akkumulators oder einer Speicherzelle wird eine Stelle nach links geschoben, wobei das Carry-Bit nach Bit 0 und Bit 7 ins Carry-Bit übernommen wird. In Abbildung 5.9 ist der Ablauf dieser Befehlsfolge aufgezeigt.



5.9 Befehlsfolge ROL, ROL INL, ROL INH

Mit ROL wird Bit 7 des Akkumulators in das Carry-Bit übertragen. Mit ROL INL übernimmt Bit 0 von INL das Carry-Bit und übergibt Bit 7 an das Carry-Bit. Das gleiche gilt für ROL INH.

Nach einem 4 maligen Durchlauf in der Schleife hat INL die 4 Bit des Akkumulators übernommen und den Inhalt der Zellen INL und INH jeweils um 4 Stellen nach links verschoben.

Dieses Programm wird jetzt von Hand assembliert. Dies verläuft genauso

wie im letzten Beispiel. Hier müssen allerdings noch die Sprungziele bestimmt werden. Vorher wird der Programmanfang auf Zelle $800 und die Adressen für den Eingabepuffer INL = $14 und INH = $15 festgelegt.

Unterprogramm PACK:

```
$800   C9 B0   PACK   CMP   #$B0
$802   30 1F          BMI    FIN
$804   C9 C6          CMP   #$C6
$806   10 1B          BPL    FIN
$808   C9 BA          CMP   #$BA
$80A   30 07          BMI   ZAHL
$80C   C9 C1          CMP   #$C1
$80E   30 13          BMI    FIN
$810   18             CLC
$811   69 09          ADC   #$09
$813   0A      ZAHL   ASL
$814   0A             ASL
$815   0A             ASL
$816   0A             ASL
$817   A0 04          LDY   #$04
$819   2A      ZAHL1  ROL
$81A   26 14          ROL    INL
$81C   26 15          ROL    INH
$81E   88             DEY
$81F   D0 F8          BNE   ZAHL1
$821   A9 00          LDA   #$00
$823   60      FIN    RTS
```

Abbildung 5.10

Bei der Berechnung der Sprungziele muß man immer beachten, daß der Befehlsfolgezähler schon auf dem nächsten auszuführenden Befehl steht. Bei der Papier und Bleistiftmethode wird man in den meisten Fällen Vorwärtssprünge nicht sofort angeben können. Die Speicherzelle, welche später diesen Byte speichern soll, markiert man durch

einen Strich, wie in Abbildung 5.11, dem Ausschnitt einer Handassemblierung.

```
Unterprogramm PACK

PACK ≙ $800 ; INL ≙ $14 ; INH ≙ $15

$800   C9 B0   PACK   CMP #$B0
$802   30 __          BMI FIN
$804   C9 C6          CMP #$C6
$806   10 __          BPL FIN
$808   C9 BA          CMP #$BA
$80A   30 07          BMI ZAHL
$80C   C9 D1          CMP #$C1
$80E   90 __          BMI FIN
$810   18             CLC
$811   69 09          ADC #$09
$813   0A      ZAHL   ASL
$814   0A             ASL
```

5.11 Ausschnitte aus einer Assemblierung von Hand

Die Sprünge zum Programmende FIN konnten noch nicht angegeben werden, dagegen aber der Sprung in Zelle $80A BMI ZAHL. Der Marke ZAHL entspricht die Adresse $813. Damit berechnet sich die Sprungweite folgendermaßen:

Vor Ausführung des Sprunges steht der Befehlszähler auf Adresse $80C. Bis zur Adresse $813 müssen 7 Byte übersprungen werden. In den Speicherzellen $80A und $80B sind somit die Bitmuster 30 07 gespeichert.

Der Rücksprung in Adresse $81F, BNE ZAHL1, wird auf die gleiche Weise bestimmt. Der Befehlsfolgezähler steht auf Adresse $821. Bis Adresse $819 muß um 7 Byte zurückgesprungen werden. Die Umwandlung der Dezimalzahl 7 in eine negative Hexadezimalzahl wird mit Abbildung 4.6 durchgeführt. Dort findet man für –7 die Hexzahl $F8.

Die Umwandlung von Dezimalzahlen in Hexadezimalzahlen für Vorwärtssprünge erfolgt in Abbildung 4.5. Als letztes Beispiel hierzu die Bestimmung der Sprungweite von BMI FIN in Adresse $802. Der Befehlsfolgezähler zeigt auf Adresse $804, das Sprungziel FIN hat die Adresse $823. Damit müssen 31 = $1F Byte übersprungen werden.

Diese Berechnung der Sprungweiten ist bei der Programmierung und Assemblierung von Hand eine der häufigsten Fehlerquellen.

```
** ZERO PAGE VARIABLES:

INL    0014 INH    0015 Z1L    0010 Z1H    0011 Z2L    0012 Z2H    0013

** ABSOLUTE VARABLES/LABELS

PRTBYT FDDA MONITO FF59 GETCHR FD35 PRHEX  FDE3 PACK   0800 ZAHL   0813
ZAHL1  0819 FIN    0823 ADD16  0824 FEHLER 083C GETWD  083D GETWD1 0843
GETWD2 0854 MAIN   0900

0800           ;
0800           ;*******************************
0800           ;*                             *
0800           ;* PACK LIEST EIN ASCII ZEI-   *
0800           ;* CHEN UND UEBERGIBT ES DEM   *
0800           ;* EINGABEPUFFER.              *
0800           ;*                             *
0800           ;*******************************
0800           ;
0800           ;
0800 C9B0      PACK   CMP #$B0
0802 301F             BMI FIN
0804 C9C6             CMP #$C6
0806 101B             BPL FIN
0808 C9BA             CMP #$BA
080A 3007             BMI ZAHL
080C C9C1             CMP #$C1
080E 3013             BMI FIN
0810 18               CLC
0811 6909             ADC #$09
0813 0A        ZAHL   ASL
0814 0A               ASL
0815 0A               ASL
0816 0A               ASL
0817 A004             LDY #$04
0819 2A        ZAHL1  ROL
081A 2614             ROL INL
```

```
081C 2615            ROL INH
081E 88              DEY
081F D0F8            BNE ZAHL1
0821 A900            LDA #$00
0823 60       FIN    RTS
0824          ;
0824          ;******************************
0824          ;*                            *
0824          ;* ADDITION VOM 2 16-BIT      *
0824          ;* ZAHLEN OHNE VORZEICHEN     *
0824          ;*                            *
0824          ;******************************
0824 18       ADD16  CLC
0825 A510            LDA Z1L
0827 6512            ADC Z2L
0829 8510            STA Z1L
082B A511            LDA Z1H
082D 6513            ADC Z2H
082F 8511            STA Z1H
0831 B009            BCS FEHLER
0833 20DAFD          JSR PRTBYT
0836 A510            LDA Z1L
0838 20DAFD          JSR PRTBYT
083B 60              RTS
083C 00       FEHLER BRK
083D          ;******************************
083D          ;*                            *
083D          ;* GETWD LIEST EIN 16-BIT     *
083D          ;* WORT IN DEN EINGANGS-      *
083D          ;* PUFFER.                    *
083D          ;*                            *
083D          ;******************************
083D A900     GETWD  LDA #$00               ;00 --> INL,INH
083F 8514            STA INL
0841 8515            STA INH
0843 2035FD   GETWD1 JSR GETCHR
0846 200008          JSR PACK               ; WIEDERHOLEN,BIS
0849 D009            BNE GETWD2             ; CHR <> HEXZEICHEN
084B A514            LDA INL                ;LETZTE 2 ZAHLEN IN AKKU
084D 290F            AND #$0F               ;VORLETZTE ZAHL AUSBLENDEN
084F 20E3FD          JSR PRHEX              ;LETZTE ZAHL AUF BILDSCHIRM
0852 10EF            BPL GETWD1
0854 60       GETWD2 RTS
0900                 ORG $900
0900          ;
0900          ;
0900          ;******************************
0900          ;*                            *
0900          ;*     HAUPTPROGRAMM          *
0900          ;*     STARTADRESSE=$900      *
0900          ;*                            *
0900          ;******************************
0900          ;
0900          ;
0900 203D08   MAIN   JSR GETWD              ;ERSTE ZAHL LESEN
0903 A514            LDA INL
0905 8510            STA Z1L
0907 A515            LDA INH
0909 8511            STA Z1H
090B 203D08          JSR GETWD              ;ZWEITE ZAHL LESEN
090E A514            LDA INL
0910 8512            STA Z2L
0912 A515            LDA INH
0914 8513            STA Z2H
0916 202408          JSR ADD16
0919 4C59FF          JMP MONITO
                     END
```

Abbildung 5.12

In Abbildung 5.12 ist das vollständige Programm zur Addition zweier 16-bit Binärzahlen angeführt.

In den Speicherzellen $800 bis $845 sind die Unterprogramme PACK, ADD16, und GETWD gespeichert. Das Hauptprogramm beginnt bei Adresse $900 und endet bei $918.

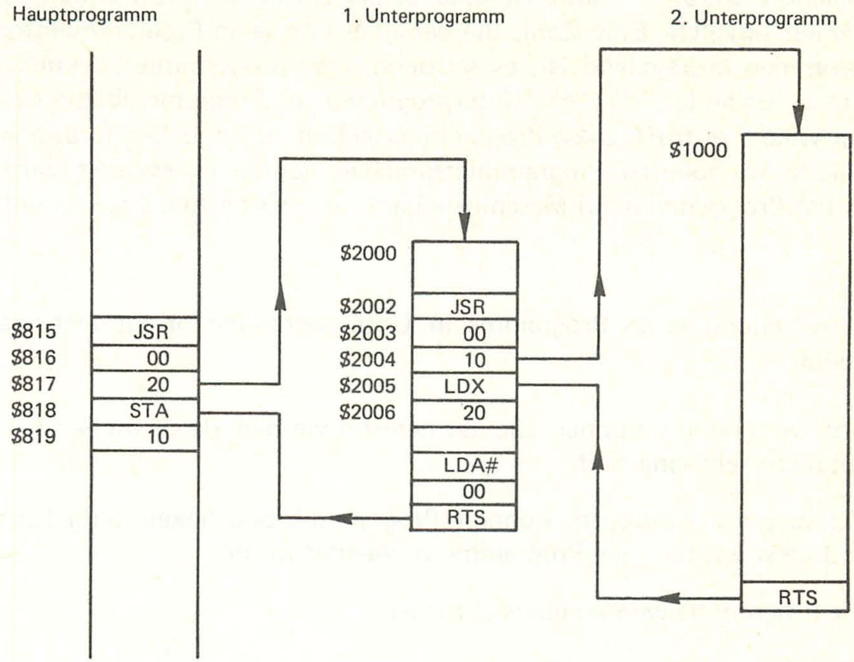
5.5 Unterprogramme

In den letzten Programmen sind immer Unterprogramme verwendet worden. Der Programmablauf: Hauptrogramm → Unterpogramm → Hauptprogramm ist in Abbildung 5.13 dargestellt.

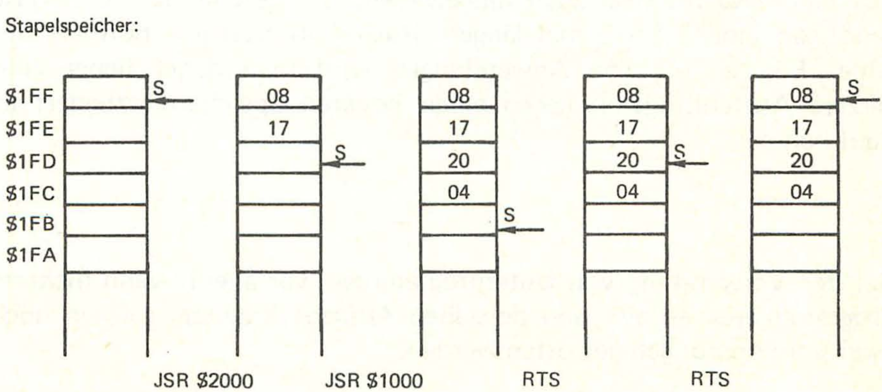
Im Hauptprogramm ist in Adresse $815 ein Unterprogrammsprung nach Adresse $2000 programmiert. In diesem Unterprogramm wird nochmals nach einem Unterpogramm, beginnend mit Adresse $1000 verzweigt (Abbildung 5.13 a). Abbildung 5.13 b zeigt den Eintrag der Rücksprungadressen in den Stapelspeicher. Wie schon erwähnt, belegt der Stapelspeicher den Adressbereich $100 bis $1FF mit $1FF als TOS (Top of Stack). Mit S ist in Abbildung 5.13 b der Stapelzeiger bezeichnet. Bei der Ausführung des JSR-Befehls im Hauptprogramm zeigt der Befehlsfolgezähler auf Adresse $818. Diese Adresse, allerdings um 1 erniedrigt wird in den Stapelspeicher mit ADL, ADH übertragen. Der Stapelzeiger, der vorher auf $1FF zeigte wird um zwei erniedrigt und zeigt nun auf Adresse $1FD. Der Befehlsfolgezähler übernimmt die Anfangsadresse des Unterprogramms: $2000.

Beim nächsten Unterprogrammsprung im 1. Unterprogramm wird als Rücksprungadresse $2005, wiederum um 1 erniedrigt, übertragen. Der Stapelzeiger zeigt danach auf Adresse $1FB. Im 2. Unterprogramm erfolgt kein weiterer Unterprogrammsprung mehr. Es wird normal mit RTS verlassen. Der Befehl RTS bewirkt, daß der Befehlsfolgezähler die Adresse $2005 übernimmt und den Stapelzeiger auf die Adresse $1FD hochsetzt. Das 1. Unterprogramm wird bis RTS durchlaufen. Jetzt wird vom Befehlsfolgezähler die Adresse $818 übernommen und der Stapelzeiger auf Adresse $1FF hochgesetzt. Damit ist der Ausgangszustand vor dem Einsprung ins 1. Unterprogramm wieder hergestellt.

a)



b)



5.13 Programmablauf mit Unterprogrammsprüngen

In den 256 Speicherzellen des Stapels können 128 Rücksprungadressen gespeichert werden. Damit ist eine Schachtelung von 128 Unterprogrammen möglich. Eine Zahl, die bei einer normalen Programmierung vollkommen ausreichend ist, es sei denn, man programmiert **rekursiv**. Rekursiv bedeutet, daß ein Unterprogramm im Programmablauf sich selbst wieder aufruft. Diese Programmiertechnik, die z. B. bei Iterationsverfahren in höheren Programmiersprachen häufig angewendet wird, wird bei Programmen auf Maschinenebene nur recht selten angewendet.

Die Aufteilung eines Programms in Unterpogramme bringt mehrere Vorteile:

1. Unterprogramme können leichter getestet werden, da sie im Normalfall nicht sehr lang sind.
2. Mit Unterprogrammen können Programm-Bibliotheken aufgebaut und sie somit für neue Programme verwendet werden.
3. Die Programme werden übersichtlicher.
4. Sie sparen an Speicherplatz.

Auf einen Nachteil sei auch hingewiesen. Die Befehle JSR und RTS benötigen eine 2 bis 3 mal längere Ausführungszeit wie normale Befehle. Für zeitkritische Anwendungen wird man daher lieber eine kürzere Ausführungszeit gegen einen höheren Speicherplatzbedarf in Kauf nehmen.

Bei der Verwendung von Unterprogrammen, vor allem, wenn mehrere Programmierer an ein- und derselben Aufgabe arbeiten, müssen noch zwei Vereinbarungen getroffen werden.

1. Wie erfolgt die Übergabe von Daten an das Unterprogramm ?
2. Werden die Registerinhalte im Hauptprogramm oder im Unterprogramm gerettet ?

Die Übergabe von Daten in ein Unterprogramm kann auf dreierlei Weise erfolgen:

1. Daten werden über die Register (Akkumulator, X-Register, Y-Register) übergeben.

2. Daten werden im Hauptprogramm im Stapelspeicher abelegt, von wo sie aus dem Unterprogramm abgerufen werden.

3. Die Daten belegen einen festen Speicherbereich, auf den Hauptprogramm und Unterprogramm zugreifen können.

Welche der 3 Möglichkeiten angewendet wird ist beliebig, sie hängt meistens von der Art des Programms ab.

In Unterprogrammen werden häufig auch die Register benötigt. Soll aber der Registerinhalt von Hauptprogramm erhalten bleiben, so muß dieser in irgendeiner Form gerettet werden. Hierbei können grundsätzlich zwei Konventionen getroffen werden.

1. Die Registerinhalte werden im Unterprogramm gerettet.

2. Die Registerinhalte werden im Hauptprogramm gerettet.

Bei der ersten Methode braucht sich der Programmierer keine Gedanken darüber zu machen, ob die Inhalte verändert werden oder nicht. Allerdings kann es sein, daß Speicherplatz verschwendet wird, weil die Register gerettet werden, obwohl es nicht notwendig war.

Diese Speicherplatzverschwendung ist bei der zweiten Methode nicht gegeben. Hier braucht der Programmierer aber in der Programmbeschreibung des Unterprogramms genaue Angaben über die verwendeten Register.

Zum Abschluß des Kapitels über Unterprogramme noch 2 Routinen zum Registerinhaltretten und -wiederherstellen und 2 Programmiertricks.

SAVE rettet die Registerinhalte auf dem Stapelspeicher,

LOAD holt sie von dort wieder zurück.

```
0800           ;
0800           ;****************************
0800           ;*                          *
0800           ;* SAVE RETTET A,X,Y-REG-   *
0800           ;* ISTER IM STAPELSPEICHER  *
0800           ;*                          *
0800           ;****************************
0800           ;
0800           ;
0800 48        SAVE    PHA                 ;<A> --> S
0801 8A                TXA
0802 48                PHA                 ;<X> --> S
0803 98                TYA
0804 48                PHA                 ;<Y> --> S
0805 60                RTS
0806           ;
0806           ;
0806           ;****************************
0806           ;*                          *
0806           ;* LOAD LADET DIE REGISTER- *
0806           ;* INHALTE WIEDER ZURUECK   *
0806           ;*                          *
0806           ;****************************
0806           ;
0806           ;
0806 68        LOAD    PLA
0807 A8                TAY                 ;<S> --> Y
0808 68                PLA
0809 AA                TAX                 ;<S> --> X
080A 68                PLA                 ;<S> --> A
080B 60                RTS
080C           ;
080C           ;
                       END
```

5.14 Unterprogramme SAVE und LOAD

Programmiertrick 1:

Es gibt normalerweise keine Möglichkeit festzustellen welchen Inhalt der Befehlsfolgezähler hat. Wie wir aber in Abbildung 5.13 b gesehen haben, wird bei einem JSR der augenblickliche Stand des Befehlsfolgezählers in den Stapelspeicher übernommen. Dort bleibt er auch nach einem RTS erhalten und kann von dort in die Register übernommen werden. Dazu braucht man noch eine Speicherzelle im Monitor, die ein RTS enthält. Dann kann man mit folgenden Programm feststellen, wo sich dieses zur Zeit im Speicher befindet.

Diese Methode wird zum Beispiel in den Programmen auf den Ein-Ausgabeplatinen des APPLE verwendet, um festzustellen, in welchem Steckplatz sich diese Platine befindet.

```
** ZERO PAGE VARIABLES:

ADL     0010  ADH     0011

0800                      DCM "PR#1"
0800             ADL      EPZ $10
0800             ADH      EPZ $11
0800             ;
0800             ;
0800             ;
0800             ;*****************************
0800             ;*                           *
0800             ;* BEFEHLSFOLGEZAEHLER NACH  *
0800             ;* ADL UND ADH.              *
0800             ;*                           *
0800             ;*****************************
0800             ;
0800             ;
0800 20BDFF               JSR $FFBD          ;RTS IM APPLE-MONITOR
0803 BA                   TSX                ; S --> X
0804 BD0001               LDA $100,X         ;<S> --> A
0807 8511                 STA ADH
0809 CA                   DEX
080A BD0001               LDA $100,X         ;<S-1> --> A
080D 8510                 STA ADL
080F 60                   RTS
                          END
```

5.15 Befehlsfolgezähler nach ADL und ADH

Eine andere Anwendung besteht darin, absolute Adressen eines Programms so umzurechnen, daß dieses Programm überall im Speicher lauffähig ist. Trifft man dazu noch die Konvention, daß ein Programm nur auf einer Seite (Adressen $1000, $1100, $1200 usw. bis $FE00) anfangen darf, so kann man durch ein Vorprogramm auch längere Programme speicherverschieblich (relocatierbar) machen.

```
0800             ;
0800             ;*****************************
0800             ;*                           *
0800             ;* AENDERUNG ABSOLUTER       *
0800             ;* ADRESSEN IM PROGRAMM      *
0800             ;*                           *
0800             ;*****************************
0800             ;
0800             ;
0800             ;
0800 20BDFF               JSR $FFBD
0803 BA                   TSX
0804 BD0001               LDA $100,X
0807 8D1008               STA M1+2
080A EA          M        NOP
080B EA                   NOP
080C EA                   NOP
080D EA                   NOP
080E 4C0A08      M1       JMP M
                          END
```

5.16 Änderung absoluter Adressen durch Programm

Die absolute Adresse des Sprungbefehls in Speicherzelle $810 wird durch das Programm so abgeändert, daß der Programmanfang dieses DEMO-Programms auf jeder Seite des Speichers liegen kann.

Programmiertrick 2:

Ist der letzte Befehl in einem Unterprogramm vor dem RTS ein JSR zu einem anderen Unterprogramm, so kann die Befehlsfolge

```
JSR UP
RTS                 durch

JMP UP
```

ersetzt werden. Das RTS im Programmteil UP führt im ersten Fall auf das RTS nach dem JSR Befehl und von dort zurück ins Hauptprogramm. Im zweiten Fall führt das RTS in UP direkt zurück ins Hauptprogramm. Dadurch wird Platz (1 Byte) und Zeit gespart.

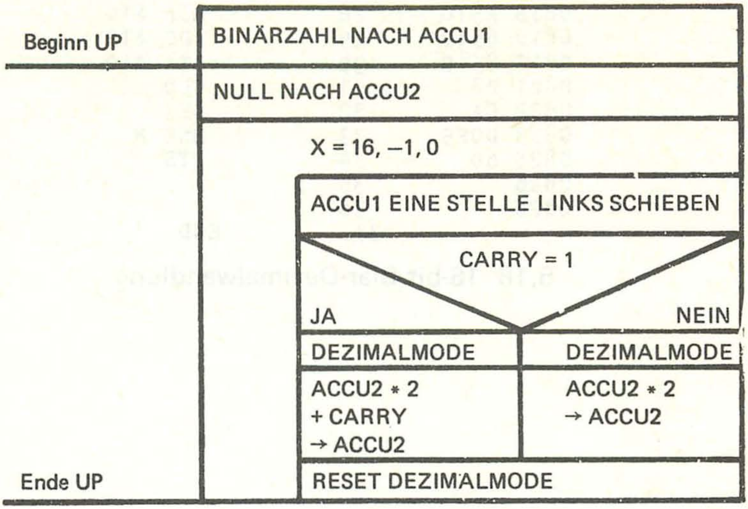
5.6 Struktogramm

Neben den Flußdiagrammen besteht noch die Möglichkeit, Programmabläufe durch ein Struktogramm zu erläutern. Diese Darstellung wollen wir für ein Programm zur Umwandlung einer 16-bit Binärzahl ohne Vorzeichen in eine 5 stellige Dezimalzahl verwenden.

Dazu treffen wir folgende Vereinbarungen:

Die Zellen $1E und $1F enthalten die 16-bit Zahl mit MSB in 1E und LSB in $1F. Diese beiden Zellen bezeichnen wir mit ACCU 1. Das 5-stellige Ergebniss wird in den Zellen $10 bis $12 abgelegt, wobei die untersten 4-bit der Zelle $10 die höchste Dezimalstelle enthalten. Diese Zellen bezeichnen wir mit ACCU2.

Die Umsetzung erfolgt dadurch, daß ACCU1 eine Stelle nach links geschoben wird und gleichzeitig der Inhalt von ACCU2 verdoppelt wird. Diese Verdoppelung geschieht in der Betriebsart Dezimal. Tritt beim Linksschieben von ACCU1 ein Übertrag ins Carry-Bit auf, so wird der Inhalt von ACCU2 um Eins erhöhet. Diese Schleife wird 16 mal durchlaufen.
(Mathematisches Verfahren: HORNER-Schema)
Das Verfahren ist in einem Struktogramm in Abbildung 5.17 gezeigt.



5.17 Struktogramm für 16-bit Binär-Dezimalwandlung

Beim Einsprung in das Unterprogramm ist die 16-bit Zahl in ACCU1. Zuerst wird ACCU2 auf Null gesetzt, dann folgt eine Schleife mit dem X-Register als Zähler. In der Schleife erfolgt das schon angegebene Verfahren zur Umsetzung. Das Programm hierzu zeigt Abbildung 5.18.

```
0800            ;
0800            ;
0800            ;******************************
0800            ;*                            *
0800            ;* 16 BIT BINAER DEZIMAL-     *
0800            ;* WANDLUNG.                  *
0800            ;*                            *
0800            ;******************************
0800            ;
0800            ;
0800            ;
0800 A900       BINBCD LDA #$00
0802 8510              STA $10
0804 8511              STA $11
0806 8512              STA $12
0808 A210              LDX #$10
080A 061F       M      ASL $1F
080C 261E              ROL $1E
080E F8                SED
080F A512              LDA $12
0811 6512              ADC $12              ;ADDITION MIT CARRY
0813 8512              STA $12
0815 A511              LDA $11
0817 6511              ADC $11
0819 8511              STA $11
```

```
081B A510          LDA $10
081D 6510          ADC $10
081F 8510          STA $10
0821 D8            CLD
0822 CA            DEX
0823 D0E5          BNE M
0825 60            RTS
0826            ;
0826            ;
                   END
```

5.18 16-bit Biär-Dezimalwandlung

Assembler-Programmierung

6. Assembler–Programmierung

Im letzten Kapitel haben wir die in Assemblersprache geschriebenen Programme mit der Papier- und Bleistift Methode in den Maschinencode übertragen. Diese Übertragung kann auch durch ein Programm, dem Assembler, durchgeführt werden. Folgende Grundelemente muß ein Assembler enthalten:

1. Einen Text-Editor zum Schreiben des Quell-Textes (Source Code). Dieser Text-Editor kann entweder Zeilenorientiert oder Bildschirmorientiert sein. Zeilenorientiert bedeutet, daß die Anweisungen wie in BASIC mit Zeilennummern versehen werden und Änderungen durch Angabe der Zeilennummer und eines Editierbefehls (Einfügen, Löschen, Ändern) durchgeführt werden. Bildschirmorientiert bedeutet, daß ein Assemblertext auf dem Bildschirm durch Cursor-Bewegungen und Editierbefehle korriegiert und verändert wird.

2. Im Text müssen für Marken und Sprungziele symbolische Bezeichnungen zulässig sein. Die Länge dieser Bezeichnungen sollte mindestens 6 Zeichen betragen.

3. Nach einer Assemblierung sollte eine Symbol-Tabelle ausgedruckt werden, die die Zuordnung zwischen symbolischen Namen und der Speicheradresse enthält. So eine Tabelle ist eine große Hilfe bei der Fehlersuche in Programmen.

Folgende Punkte erleichtern die Arbeit des Programmierens, sind aber

nicht unbedingt erforderlich:

1. Programmteile auf Diskette oder Kassette können an beliebigen Stellen im Text durch den Editor eingefügt werden.

2. Tritt in einem Assemblertext eine Befehlsfolge mehrmals auf, so soll diese durch einen selbsdefinierten Befehl zu einem MAKRO zusammengefasst werden können. Dieser Name, bestehend aus 3 Buchstaben (z. B. POP, PUL) wird dann im Text wie ein Assemblerbefehl aufgeführt und bei der Übersetzung dann durch die Befehlsfolge ersetzt (siehe 8.1 Makro-Assembler des PASCAL-Systems).

Im übrigen gilt: Je komfortabler der Assembler, desto einfacher die Programmentwicklung.

Bei der Programmierung mit der Papier- und Bleistiftmethode haben wir die Anfangsadresse des Programms und die Adressen von Speicherzellen durch einfache Zuordnung festgelegt. Diese Festlegung muß auch bei der Übersetzung durch ein Programm getroffen werden. Hierzu werden sogenannte Pseudobefehle verwendet. Dies sind Befehle, die der Assembler zum Übersetzen des Programms benötigt, die aber im Programmablauf nicht auftretten. Mit diesen Pseudo-Befehlen wollen wir uns zunächst beschäftigen. Leider gibt es Unterschiede in der Schreibweise bei den einzelnen 6502 Assemblern. Wir werden diese bei der Besprechung der Assembler in Kapitel 8 noch einzeln aufführen.

6.1 Assembler-Pseudo-Befehle

1. **OBJ**
   Schreibweise: OBJ ⟨AUSDRUCK⟩
   Mit diesem Pseudo-Befehl wird die Anfangsadresse des übersetzten Programms (dem Object-Code) festgelegt. Diese Adresse muß **nicht** gleich die Startadresse des Programms sein. Hier wird nur angegeben, wohin das übersetzte Programm gespeichert wird.

   Beispiel: OBJ $A00
   Das übersetzte Programm beginnt in Zelle $A00

2. **ORG**

Schreibweise: ORG ⟨ AUSDRUCK ⟩

Mit ORG (Origin) wird die Startadresse des übersetzten Programms festgelegt. Auf diese Adresse beziehen sich alle absoluten Adressangaben (Basisadresse). Die Angabe von ORG beinhaltet immer, wenn nicht anders angegeben, den Befehl OBJ.

Beispiel 1: ORG $800

Basisadresse des übersetzten Programms ist $800, der übersetzte Code beginnt ebenfalls mit dieser Adresse.

Beispiel 2: ORG $800
OBJ $1000

Die Basisadresse des übersetzten Programms ist $800. Der Code ist allerdings ab Adresse $1000 gespeichert. Soll dieses Programm ausgeführt werden, so muß der Code durch einen MOVE-Befehl mit dem Monitor an die Adresse $800 umgespeichert werden.

3. **EQU**

Schreibweise: ⟨ NAME ⟩ EQU ⟨ AUSDRUCK ⟩

Mit EQU (Equate) erfolgt die Zuordnung eines symbolischen Namens zu einer Adresse.

Beispiel: ANFANG EQU $2000

Der symbolische Name ANFANG entspricht der Adresse $2000. Der Assemblerbefehl JSR ANFANG bedeutet damit einen Sprung in ein Unterprogramm, daß bei der Adresse $2000 beginnt. Einige Assembler unterscheiden bei der EQU Definition, ob im ⟨ AUSDRUCK ⟩ eine 1 Byte oder eine 2 Byte Adresse angegeben wird. Bei der Angabe von nur 1 Byte ist damit eine Speicherzelle in der Zero-Page angewählt. Andere Assembler verlangen für die Zuordnung eines symbolischen Namens zu einer Adresse in der Zero-Page einen besonderen Pseudo-Befehl EPZ (Equate Page Zero).

Schreibweise: ⟨ NAME ⟩ EPZ ⟨ AUSDRUCK ⟩.

4. **ASC**

Schreibweise: ASC ⟨ "ZEICHENKETTE" ⟩

Mit dem Pseudo-Befehl ASC wird die folgende Zeichenkette als ASCII-Zeichen gespeichert. Damit wird auf einfache Weise Text, den man für eine interaktive Programmierung benötigt in das Pro-

gramm eingebaut.

Besipiel: ASC "NOCH WEITERE EINGABEN (J/N)?"

5. **HEX**

Schreibweise: HEX ⟨ HEX-ZEICHEN ⟩

Der Pseudo-Befehl HEX speichert Bitmuster ab, die im Programm benötigt werden.

Besipiel: HEX 00AAFF55

Als Zahlenwerte sind nur die HEX-Zahlen 0 - 9 und A - F erlaubt.

6. **END**

Schreibweise: END

Dies ist die Anweisung an den Assembler, daß das Programm an dieser Stelle zu Ende ist. Ein fehlendes END führt in allen Fällen zu einer Fehlermeldung.

Dies waren einige der Pseudo-Befehle, die als Steueranweisungen für die Assembler dienen.

Auf die Möglichkeit, Adressrechnungen durchzuführen, soll noch kurz eingegangen werden.

Durch die EQU Anweisung oder durch eine Marke wird einem symbolischen Namen eine Adresse zugeordnet, z. B.

NAME EQU $2E

Im Adressteil eines Assemblerbefehls ist es nun erlaubt, eine Adressrechnung durchzuführen. Die Anweisung LDA NAME + 1 holt den Inhalt der Speicherzelle $2F oder STA NAME – $10 speichert den Inhalt des Akkumulators nach $1E.

In den meisten Fällen wird die Addition und Subtraktion für Adressrechnungen ausreichen, obwohl einige Assembler auch Multiplikation und Division zulassen.

Eine Anweisung

LDA NAME + 1 * 3

wird von links nach rechts entwickelt, also erst NAME + 1 und dann dieses Ergebnis mal 3.

Vorsicht ist mit Adressrechnungen bei Verzweigungen geboten. Die Anweisung

BEQ + 2

bedeutet nicht, daß 2 Assembleranweisungen sondern 2 Speicherplätze weitergesprungen wird. Die folgende Befehlsfolge ist falsch:

```
      LDA  $1FFF
      BEQ  MARKE + 2
MARKE LDA  $1000
      STA  $1FFF
      DEX
```

Die Verzweigung BEQ MARKE + 2 führt nicht auf den Assemblerbefehl DEX sondern auf das 2. Adressbyte des Befehls MARKE LDA $1000. Richtig lautet dieser Programmteil:

```
      LDA  $1FFF
      BEQ  MARKE + 6
MARKE LDA  $1000
      STA  $1FFF
      DEX
```

wenn das Programm mit DEX weitergeführt werden soll.

Ferner ist bei der EQU-Anweisung noch zu beachten, daß dem Namen eine Zahl zugewiesen wird, die als Adresse interpretiert wird. Mit der Zuordnung

NAME EQU $2E

wird mit LDA NAME der Inhalt der Zelle $2E
mit LDA #NAME die Hexzahl $2E in den Akkumulator übernommen.

Nach den Programmbeispielen in Kapitel 7 werden wir auf die wichtigsten 6502 Systeme mit ihren Assemblern eingehen.

Noch einige allgemeine Bemerkungen zu Assemblern.
Führt ein Assembler eine Übersetzung durch, so ist dies im allgemeinen mit einem Durchlauf nicht getan. Es werden mehrere Durchläufe (PASS) benötigt. Die meisten Assembler benötigen 3 Durchläufe.

Im ersten Durchgang werden die Befehle entschlüsselt, die effektiven Adressen von Marken und Namen bestimmt und in einer Symbol-Tabelle angelegt.

Im zweiten Durchgang werden die symbolischen Namen durch die Adressen ersetzt.

Eine Druckausgabe auf Bildschirm oder Drucker erfolgt dann im 3. Durchlauf.

Solche Assembler bezeichnet man auch als 3 PASS ASSEMBLER.

Programm-beispiele

7. Programmbeispiele

In diesem Kapitel wollen wir nun einzelne Befehle und Adressierungsarten durch Programmbeispiele erläutern. Dafür werden Beispiele gewählt, die in dieser Form sehr oft als Unterprogramme in größeren Programmen vorkommen.

7.1 Zeitschleifen

Zeit- oder Warteschleifen werden durch Abzählen einer Zahl, die entweder in den Indexregistern oder in einer Speicherzelle abgelegt ist, gebildet.

```
0800           ;
0800           ;
0800           ;******************************
0800           ;*                            *
0800           ;* ZEITSCHLEIFE MIT X-REG.    *
0800           ;*                            *
0800           ;******************************
0800           ;
0800           ;
0800 A2AA      ZSLX     LDX #$AA
0802 CA        ZSLX1    DEX
0803 D0FD               BNE ZSLX1
0805 60                 RTS
0806           ;
0806           ;
0806           ;
0806           ;******************************
0806           ;*                            *
0806           ;* ZEITSCHLEIFE MIT X- UND Y-*
0806           ;* REGISTER WERTE WERDEN BEIM*
0806           ;* EINSPRUNG UEBERGEBEN       *
0806           ;*                            *
0806           ;******************************
0806           ;
0806           ;
0806 CA        ZSLXY    DEX
0807 D0FD               BNE ZSLXY
0809 88                 DEY
080A D0FA               BNE ZSLXY
080C 60                 RTS
080D           ;
080D           ;
```

```
080D          ;
080D          ;******************************
080D          ;*                            *
080D          ;* BILDUNG EINER ZEITSCHLEIFE*
080D          ;* DURCH AUFZAEHLEN.          *
080D          ;*                            *
080D          ;******************************
080D          ;
080D          ;
080D          Z1L      EPZ $10
080D          Z1H      EPZ Z1L+1
080D          ;
080D A900     ZSL      LDA #$00
080F AA                TAX
0810 A8                TAY
0811 C411     ZSL1     CPY Z1H
0813 D004              BNE ZSL2
0815 E410              CPX Z1L
0817 F007              BEQ ZSLRTS
0819 E8       ZSL2     INX
081A D0F5              BNE ZSL1
081C C8                INY
081D 18                CLC
081E 90F1              BCC ZSL1
0820 60       ZSLRTS   RTS
                       END
```

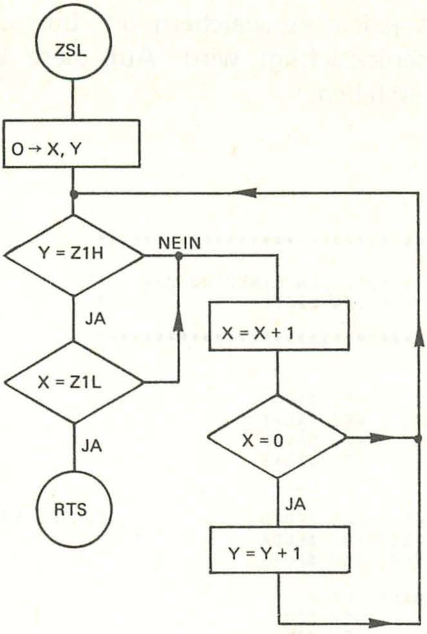
7.11 Programme Zeitschleifen

Im ersten Programm ZSLX wird eine 8-bit Zahl $AA in das X-Register übernommen. Von dieser Zahl wird solange Eins abgezogen, bis der Inhalt des X-Registers Null ist.

Im zweiten Programm wird an die beiden Register eine 16-bit Zahl vor dem Einsprung in die Schleife übergeben. Dabei enthält das Y-Register das MSB und das X-Register das LSB. Diese 16-bit Zahl wird solange um Eins erniedrigt, bis der Inhalt beider Register Null ist.

Im dritten Beispiel wird der Inhalt der beiden Register solange um Eins erhöht, bis er gleich einer 16-bit Zahl ist, die in Z1L und Z1H gespeichert ist. Das Flußdiagramm dieses Unterprogramms zeigt Abbildung 7.12. Zu diesem Programm ein Hinweis:

Der Rücksprung zu ZSL1 nach dem Erhöhen des Y-Registers erfolgt mit der Befehlsfolge CLC, BCC ZSL1+1. Dies entspricht einem JMP-Befehl. Die Verwendung eines JMP-Befehls an dieser Stelle hätte aber die Einführung einer absoluten Adresse bedeutet (JMP ZSL1+1), das Programm wäre somit nicht mehr speicherverschieblich gewesen. Mit der angegebenen Befehlsfolge bleibt dies aber erhalten.



7.12 Flußdiagramm Aufzählen

7.2 Tabellen und Listen

Die nächsten Programme beziehen sich auf Tabellen und Listen. Folgende Programmteile sollen beschrieben werden. Aufstellen von Tabellen, Einfügen, Löschen und Vergleichen von Elementen, Listen sortieren und Listen umspeichern.

7.21 Aufstellen einer unsortierten Liste

Im Programm ist SAL, SAH die Anfanggsadresse der Liste und EAL, EAH die Adresse des zuletzt gespeicherten Eintrags. Die Eingabe von Elementen wird durch die Eingabe des Steuerzeichens CTRL–C (ETX) beendet.

Zum Abspeichern des eingegebenen Wertes wird die indirekt indizierte Adressierung verwendet (Zeile 38 in Abbildung 7.21). Der Inhalt des Y-Registers ist aber dabei Null, so daß diese Form der Adressierung einer indirekten Adressierung entspricht. Der Inhalt

von EAL wird nach jedem Abspeichern um Eins erhöht, wobei ein Übertrag in EAH berücksichtigt wird. Auf diese Weise lassen sich beliebig lange Listen erstellen.

```
0804            ;
0804            ;
0804            ;*****************************
0804            ;*                           *
0804            ;* AUFSTELLEN EINER UNSOR-   *
0804            ;* TIERTEN LISTE             *
0804            ;*                           *
0804            ;*****************************
0804            ;
0804            ;
0804            SAL    EPZ $10
0804            SAH    EPZ SAL+1
0804            EAL    EPZ SAL+2
0804            EAH    EPZ SAL+3
0804            ;
0804            ;
0804            COUT   EQU $FDED                 ;AUSGABE EINES ASCII ZEICHENS
0804            PRTBYT EQU $FDDA
0804            GETCH  EQU $FD35
0804            ;
0804 A9BA       TABANF LDA #":"
0806 20EDFD            JSR COUT
0809 A900              LDA #$00
080B 8510              STA SAL
080D 8512              STA EAL
080F A910              LDA #$10
0811 8511              STA SAH
0813 8513              STA EAH
0815 2035FD     TAB1   JSR GETCH
0818 C983              CMP #$83                  ;ETX = CTRL-C
081A D001              BNE TAB2
081C 60                RTS
081D 20EDFD     TAB2   JSR COUT
0820 A000              LDY #$00
0822 9112              STA (EAL),Y
0824 E612              INC EAL
0826 D002              BNE TAB3
0828 E613              INC EAH
082A A9A0       TAB3   LDA #" "
082C 20EDFD            JSR COUT
082F 18                CLC
0830 90E3              BCC TAB1
                       END
```

7.21 Aufstellen einer unsortierten Liste

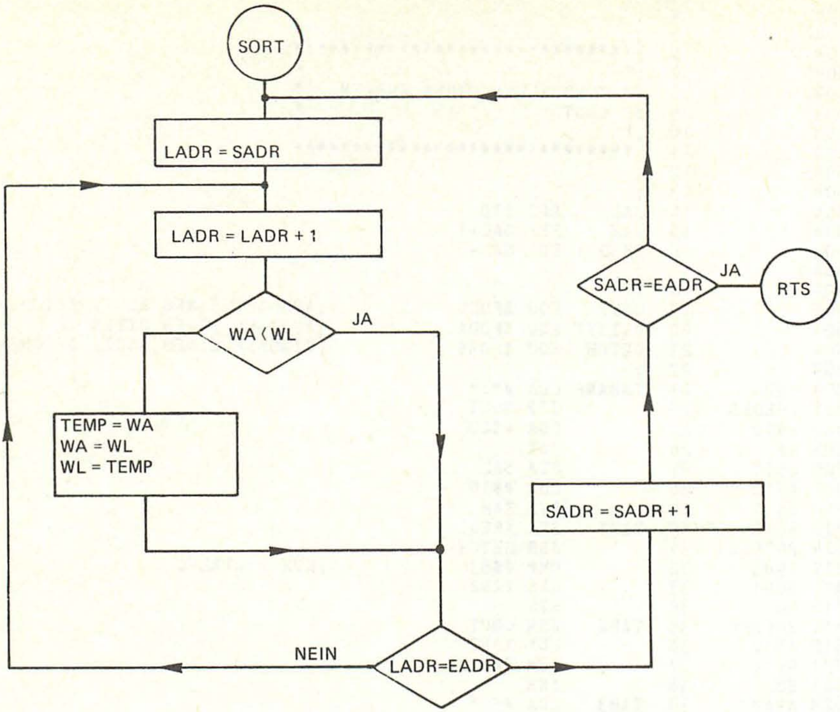
Für eine kurze Liste (Zahl der Elemente kleiner 256) kann man ein Programm nach Abbildung 7.22 verwenden. Hier erfolgt eine einfache Indizierung mit dem X-Register, deshalb muß der jeweilige Inhalt von X in einer Hilfszelle XREG gespeichert werden.

```
0804             ;
0804             ;
0804             ;*****************************
0804             ;*                           *
0804             ;* AUFSTELLEN EINER KURZEN   *
0804             ;* LISTE                     *
0804             ;*                           *
0804             ;*****************************
0804             ;
0804             ;
0804             SAL     EPZ $10
0804             SAH     EPZ SAL+1
0804             XREG    EPZ SAL+2
0804             ;
0804             ;
0804             COUT    EQU $FDED              ;AUSGABE EINES ASCII ZEICHENS
0804             PRTBYT  EQU $FDDA              ;AUSGABE EINES BYTES
0804             GETCH   EQU $FD35              ;EINGABE EINES ASCII ZEICHENS
0804             ;
0804 A9BA        TABANF  LDA #":"
0806 20EDFD              JSR COUT
0809 A900                LDA #$00
080B AA                  TAX
080C 8510                STA SAL
080E A910                LDA #$10
0810 8511                STA SAH
0812 8612        TAB1    STX XREG
0814 2035FD              JSR GETCH
0817 C983                CMP #$83               ;ETX = CTRL-C
0819 D001                BNE TAB2
081B 60                  RTS
081C 20EDFD      TAB2    JSR COUT
081F A612                LDX XREG
0821 9510                STA SAL,X
0823 E8                  INX
0824 A9A0        TAB3    LDA #" "
0826 20EDFD              JSR COUT
0829 18                  CLC
082A 90E6                BCC TAB1
                        END
```

7.22 Aufstellen einer kurzen Liste

7.22 Sortieren einer Liste

Zum Sortieren einer Liste wird das Bubblesort-Verfahren verwendet. Dabei wird das erste Element einer Liste mit den folgenden Elementen verglichen. Wird ein kleineres Element gefunden, so wird dieses an die erste Stelle gesetzt. Der Vergleich wird bis ans Ende der Liste weitergeführt. Nach diesem ersten Durchgang ist das oberste Element der Liste das kleinste vorkommende Element. Somit kann der 2. Durchlauf mit dem Vergleich des zweiten Elementes mit den Folgeelementen beginnen. Die Liste ist um einen Eintrag verkürzt worden. Das Verfahren ist beendet wenn die Startadresse (SADR) gleich der Endadresse (EADR) ist. Im Flußdiagramm (Abbildung 7.23) bezeichnen weiterhin LADR die laufende Adresse, WA den Wert des Elementes am Anfang der Liste und WL den Wert des laufenden Elementes.



7.23 Flußdiagramm BUBBLESORT

Im Programm BUBBLESORT in Abbildung 7.24 werden für die Vergleiche und die Adressrechnungen Unterprogramme verwendet. Weiterhin wird in diesem Unterprogramm die Anfangsadresse der Liste verändert. Wird diese später noch benötigt, so muß sie außerhalb des Programmteils gerettet werden.

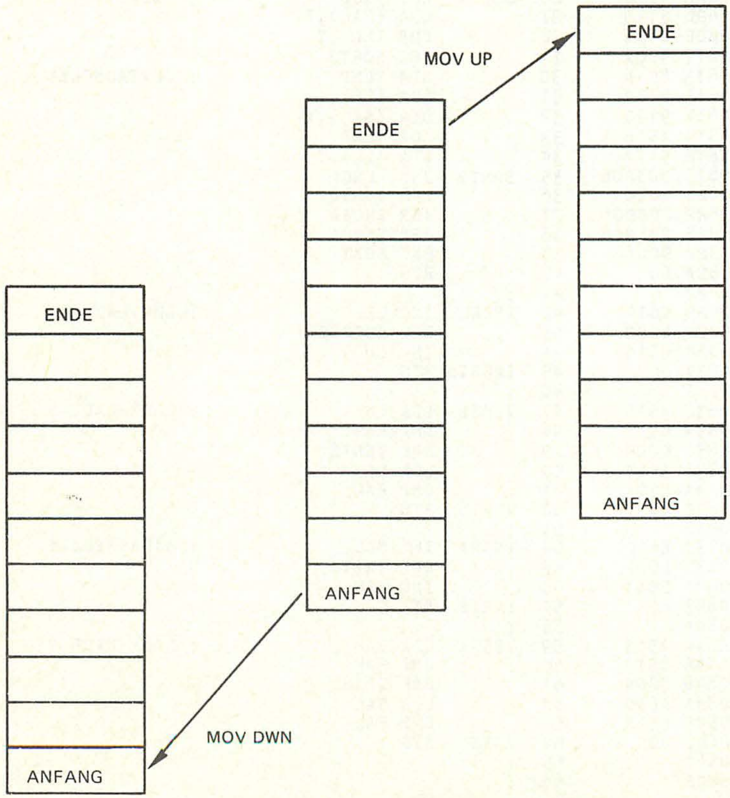
Besondere Beachtung muß man bei solchen Programmen auf die Vergleiche legen. Dabei ist es nicht gleichgültig, an welcher Stelle der Laufparameter erhöht wird. Der Vergleich der Elemente muß bis zum letzten Element durchgeführt werden. Der Adressvergleich findet am Ende der Schleife statt. Das Sortierverfahren ist beendet, wenn das vorletzte Element mit dem letzten Element verglichen wurde. Deshalb findet der Vergleich SADR = EADR gleich am Anfang dieser Schleife, nach dem Erhöhen von SADR statt.

```
0800          ;
0800          ;
0800          ;*****************************
0800          ;*                           *
0800          ;* SORTIEREN EINER LISTE     *
0800          ;* BUBBLESORT                *
0800          ;*                           *
0800          ;*****************************
0800          ;
0800          ;
0800          SAL    EPZ $10
0800          SAH    EPZ SAL+1
0800          EAL    EPZ SAL+2
0800          EAH    EPZ SAL+3
0800          LL     EPZ SAL+4
0800          LH     EPZ SAL+5
0800          TEMP   EPZ SAL+6
0800          ;
0800          ;
0800 A510     SORT   LDA SAL
0802 8514            STA LL
0804 A511            LDA SAH
0806 8515            STA LH
0808 202B08   SORT0  JSR INCRL
080B A000     SORT1  LDY #$00           ; WA=WL ?
080D B110            LDA (SAL),Y
080F D114            CMP (LL),Y
0811 900A            BCC SORT2
0813 8516            STA TEMP           ; VERTAUSCHEN
0815 B114            LDA (LL),Y
0817 9110            STA (SAL),Y
0819 A516            LDA TEMP
081B 9114            STA (LL),Y
081D 203208   SORT2  JSR VERGL
0820 D0E6            BNE SORT0
0822 203D08          JSR INCRA
0825 204408          JSR VERGA
0828 D0D6            BNE SORT
082A 60              RTS
082B          ;
082B E614     INCRL  INC LL             ;LADR=LADR+1
082D D002            BNE INCRTS
082F E615            INC LH
0831 60       INCRTS RTS
0832          ;
0832 A515     VERGL  LDA LH             ; LADR=EADL ?
0834 C513            CMP EAH
0836 D004            BNE VERTS
0838 A514            LDA LL
083A C512            CMP EAL
083C 60       VERTS  RTS
083D          ;
083D E610     INCRA  INC SAL            ; SADR=SADR+1
083F D002            BNE INRTS
0841 E611            INC SAH
0843 60       INRTS  RTS
0844          ;
0844 A511     VERGA  LDA SAH            ; SADR=EADR ?
0846 C513            CMP EAH
0848 D004            BNE VRTS
084A A510            LDA SAL
084C C512            CMP EAL
084E 60       VRTS   RTS
084F          ;
084F          ;
                     END
```

7.24 Programm BUBBLESORT

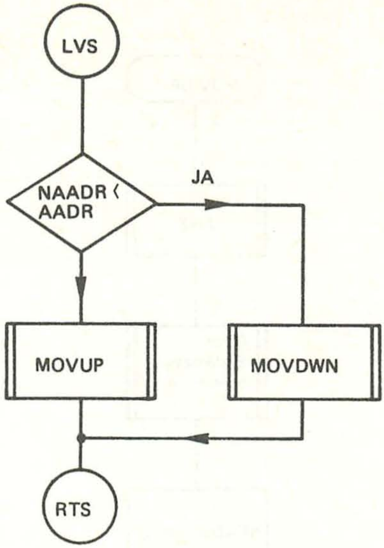
7.23 Listen verschieben

Sehr oft muß man in einem Programm Daten (Listen, Tabellen oder auch Programme) im Speicher verschieben. Insbesondere Betriebssysteme (Monitore) benötigen solche Verschiebeprogramme. Aber nicht alle Programme lassen zu, daß sich die Lage der verschobenen Liste mit der Lage der nicht verschobenen Liste überschneidet. Das Programm in Abbildung 7.29 "Listen verschieben" lässt dies allerdings zu. Dazu benötigt man die Entscheidung ob, das Umspeichern mit dem ersten oder letzten Element der Liste beginnen soll. Ist die neue Anfangsadresse (NAADR) größer als die alte Anfangsadresse (AADR), so muß die Liste mit dem letzten Element beginnend umsortiert werden. Ist die neue Anfangsadresse kleiner als die alte, so wird beim ersten Element begonnen (Abbildung 7.25).



7.25 Abbildung Listen verschieben

Dazu benötigt man zwei Unterprogramme MOVUP und MOVDWN (Abbildung 7.26).



7.26 Flußdiagramm LISTEN VERSCHIEBEN

Im Programm sind folgende symbolische Namen eingeführt worden:

AADR	≙	alte Anfangsadresse
EADR	≙	alte Endadresse
NAADR	≙	neue Anfangsadresse

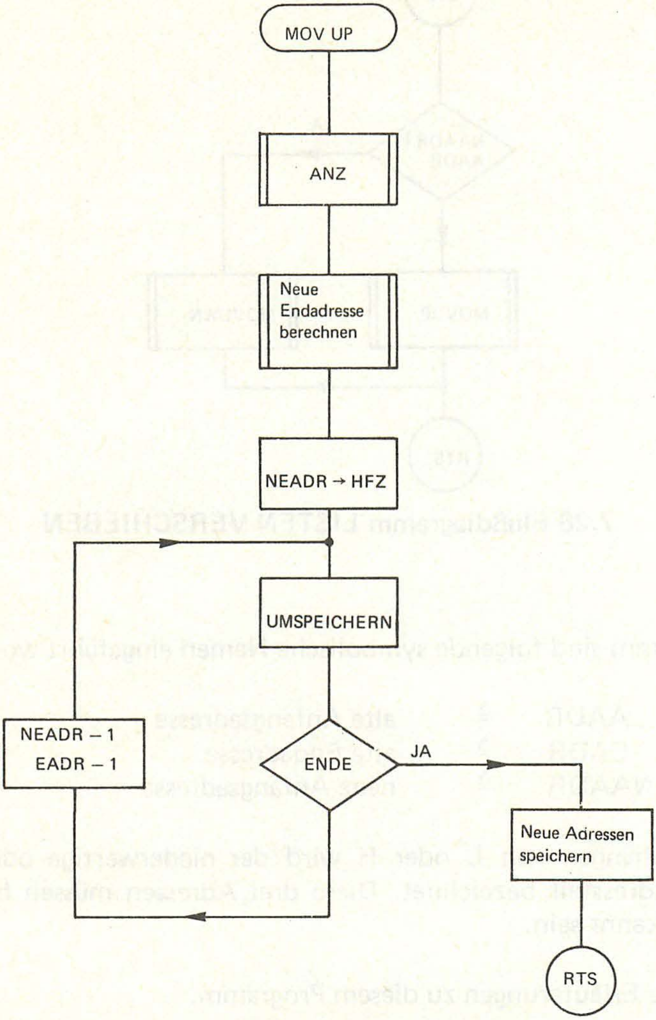
Durch Anhängen von L oder H wird der niederwertige oder höherwertige Adressteil bezeichnet. Diese drei Adressen müssen beim Einsprung bekannt sein.

Nun einige Erläuterungen zu diesem Programm.

Im Hauptprogramm findet zuerst ein Vergleich der höherwertigen Adressbytes der Anfangsadressen statt. Wenn die alte Anfangsadresse kleiner als die neue Anfangsadresse ist, so ist die Bedingung BCC MU erfüllt (siehe Abbildung 3.1).

Sind die Adressen gleich, so müssen noch die niederwertigen Adress-

bytes verglichen werden. Mit BCS MD wird dann verzweigt, wenn die alte Anfangsadresse größer ist, als die neue.



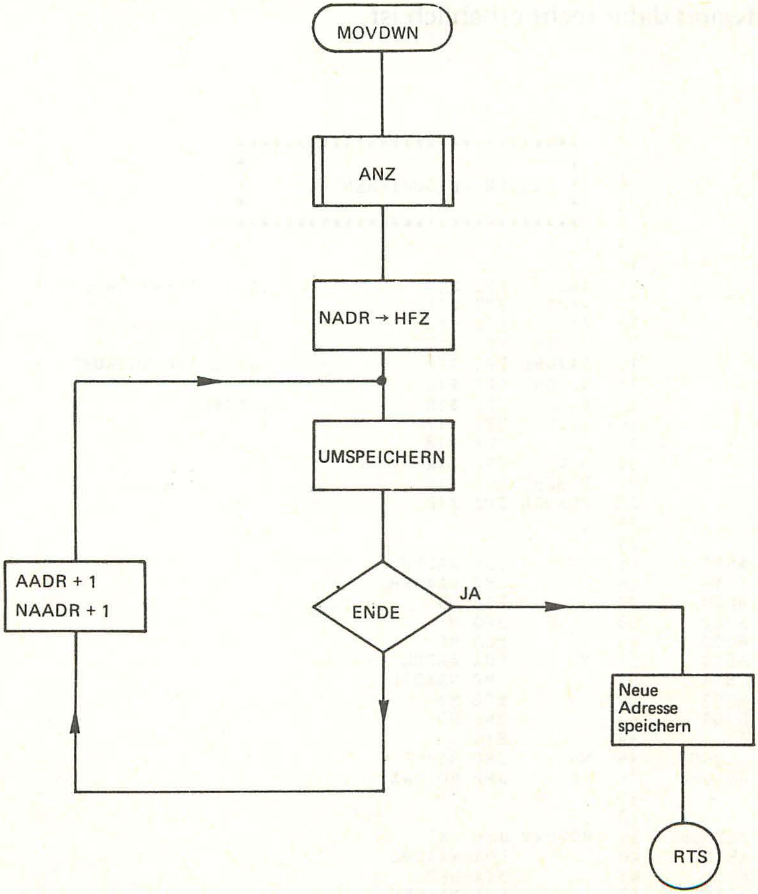
7.27 Flußdiagramm MOVUP

Allgemein gilt für Verzweigungen:

Bedingung: A ⟨ M ; Verzweigung mit BCC M

Bedingung: A ≦ M ; Verzweigung mit BCC M
BEQ M
Bedingung: A = M ; Verzweigung mit BEQ M
Bedingung: A ≧ M ; Verzweigung mit BCS M
Bedingung: A 〉M ; Verzweigung mit BEQ M1
BCS M

Das Unterprogramm MINUS dekrementiert eine 16-bit Zahl durch Subtraktion. Dabei wird gezeigt, wie Daten an ein Unterprogramm übergeben werden können. Die Zahl, die dekrementiert werden soll, muß vorher in die Hilfszellen SAVE und SAVE+1 gespeichert werden.



7.28 Flußdiagramm MOVDWN

Das Ergebnis kann dann von dort ins Hauptprogramm übernommen werden. Dieses Dekrementieren kann auch durch den Dekrementbefehl erfolgen, wie z. B. in den Zeilen 135 bis 140. Dabei muß man aber beachten, daß ein "Borgen" im höherwertigen Datenbyte dann erfolgt, wenn der Inhalt des niederwertigen Datenbytes $FF ist. Deshalb der Vergleich mit $FF in Zeile 137. Da ein Vergleich immer zwischen Akkumulator und Speicherzelle durchgeführt wird, muß der Inhalt von NEADRL vorher dorthin übernommen werden.

Die Flußdiagramme für die beiden Unterprogramme MOVUP und MOVDWN sind in den Abbildungen 7.27 und 7.28 dargestellt. Es zeigt sich, wenn man mit einem 8-bit System 16-bit Adressen berechnet daß der Aufwand dafür recht erheblich ist.

```
0800             2  ;
0800             3  ;
0800             4  ;****************************
0800             5  ;*                          *
0800             6  ;* LISTEN VERSCHIEBEN       *
0800             7  ;*                          *
0800             8  ;****************************
0800             9  ;
0800            10  ;
0800            11  AADRL  EPZ $10              ;ALTE ANFANGSADRESSE
0800            12  AADRH  EPZ $11
0800            13  EADRL  EPZ $12              ;ALTE ENDADRESSE
0800            14  EADRH  EPZ $13
0800            15  NAADRL EPZ $14              ;NEUE ANFANGSADRESSE
0800            16  NAADRH EPZ $15
0800            17  ANZL   EPZ $16              ;ANZAHL
0800            18  ANZH   EPZ $17
0800            19  HFZ    EPZ $18
0800            20  SAVE   EPZ $1A
0800            21  NEADRL EPZ $1C
0800            22  NEADRH EPZ $1D
0800            23  ;
0800            24  ;
0800 A511       25  MOVE       LDA AADRH
0802 C515       26             CMP NAADRH
0804 900D       27             BCC MU
0806 F002       28             BEQ M
0808 B00C       29             BCS MD
080A A510       30  M          LDA AADRL
080C C514       31             CMP NAADRL
080E 9003       32             BCC MU
0810 D004       33             BNE MD
0812 60         34             RTS
0813 4C9608     35  MU         JMP MOVUP
0816 4C1908     36  MD         JMP MOVDWN
0819            37  ;
0819            38  ;
0819 205608     39  MOVDWN JSR ANZ
081C A514       40             LDA NAADRL
081E 8518       41             STA HFZ
0820 A515       42             LDA NAADRH
0822 8519       43             STA HFZ+1        ;NEUE ANFANGADRESSE RETTEN
0824            44  ;
0824 A000       45  MO1        LDY #$00
```

```
0826 B110          LDA (AADRL),Y      ;UMSPEICHERN
0828 9114          STA (NAADRL),Y
082A E610          INC AADRL          ;NEUE ADRESSEN BERECHNEN
082C D002          BNE MO2
082E E611          INC AADRH
0830 E614   MO2    INC NAADRL
0832 D002          BNE MO3
0834 E615          INC NAADRH
0836        ;
0836 207F08 MO3    JSR ENDE           ; NAECHSTE ADRESSE BERECHNEN
0839 D0E9          BNE MO1
083B A518          LDA HFZ
083D 8510          STA AADRL
083F A519          LDA HFZ+1
0841 8511          STA AADRH
0843 A514          LDA NAADRL
0845 8512          STA EADRL
0847 A515          LDA NAADRH
0849 8513          STA EADRH
084B C612          DEC EADRL
084D A512          LDA EADRL
084F C9FF          CMP #$FF
0851 D002          BNE MORTS
0853 C613          DEC EADRH
0855 60     MORTS  RTS
0856        ;
0856        ;
0856 38     ANZ    SEC
0857 A512          LDA EADRL
0859 E510          SBC AADRL
085B 8516          STA ANZL
085D A513          LDA EADRH
085F E511          SBC AADRH
0861 8517          STA ANZH
0863 E616          INC ANZL
0865 D002          BNE ANZRTS
0867 E617          INC ANZH
0869 60     ANZRTS RTS
086A        ;
086A A51A   MINUS  LDA SAVE           ; DECREMENT DURCH SUBTRAKTION
086C 38            SEC
086D E901          SBC #$01
086F 851A          STA SAVE
0871 A51B          LDA SAVE+1
0873 E900          SBC #$00
0875 851B          STA SAVE+1
0877 60            RTS
0878        ;
0878 A51B   ZERO   LDA SAVE+1
087A D002          BNE ZRTS
087C A51A          LDA SAVE
087E 60     ZRTS   RTS
087F        ;
087F A516   ENDE   LDA ANZL
0881 851A          STA SAVE
0883 A517          LDA ANZH
0885 851B          STA SAVE+1
0887 206A08        JSR MINUS
088A A51A          LDA SAVE
088C 8516          STA ANZL
088E A51B          LDA SAVE+1
0890 8517          STA ANZH
0892 207808        JSR ZERO
0895 60            RTS
0896        ;
0896        ;
0896 205608 MOVUP  JSR ANZ
0899 A516          LDA ANZL
089B 851A          STA SAVE
```

```
089D A517            LDA ANZH
089F 851B            STA SAVE+1
08A1 206A08          JSR MINUS
08A4 A514            LDA NAADRL
08A6 18              CLC
08A7 651A            ADC SAVE
08A9 851C            STA NEADRL
08AB 8518            STA HFZ
08AD A51B            LDA SAVE+1
08AF 6515            ADC NAADRH
08B1 851D            STA NEADRH
08B3 8519            STA HFZ+1
08B5 A000      MOV1  LDY #$00
08B7 B112            LDA (EADRL),Y
08B9 911C            STA (NEADRL),Y
08BB C612            DEC EADRL
08BD A512            LDA EADRL
08BF C9FF            CMP #$FF
08C1 D002            BNE MOV2
08C3 C613            DEC EADRH
08C5 C61C      MOV2  DEC NEADRL
08C7 A51C            LDA NEADRL
08C9 C9FF            CMP #$FF
08CB D002            BNE MOV3
08CD C61D            DEC NEADRH
08CF 207F08    MOV3  JSR ENDE
08D2 D0E1            BNE MOV1
08D4           ;
08D4 A514            LDA NAADRL
08D6 8510            STA AADRL
08D8 A515            LDA NAADRH
08DA 8511            STA AADRH
08DC A518            LDA HFZ
08DE 8512            STA EADRL
08E0 A519            LDA HFZ+1
08E2 8513            STA EADRH
08E4 60              RTS
08E5           ;
                    END
```

7.29 Programm LISTEN VERSCHIEBEN

```
0800- A5 11 C5 15 90 0D F0 02
0808- B0 0C A5 10 C5 14 90 03
0810- D0 04 60 4C 96 08 4C 19
0818- 08 20 56 08 A5 14 85 18
0820- A5 15 85 19 A0 00 B1 10
0828- 91 14 E6 10 D0 02 E6 11
0830- E6 14 D0 02 E6 15 20 7F
0838- 08 D0 E9 A5 18 85 10 A5
0840- 19 85 11 A5 14 85 12 A5
0848- 15 85 13 C6 12 A5 12 C9
0850- FF D0 02 C6 13 60 38 A5
0858- 12 E5 10 85 16 A5 13 E5
0860- 11 85 17 E6 16 D0 02 E6
0868- 17 60 A5 1A 38 E9 01 85
0870- 1A A5 1B E9 00 85 1B 60
0878- A5 1B D0 02 A5 1A 60 A5
0880- 16 85 1A A5 17 85 1B 20
0888- 6A 08 A5 1A 85 16 A5 1B
0890- 85 17 20 78 08 60 20 56
0898- 08 A5 16 85 1A A5 17 85
08A0- 1B 20 6A 08 A5 14 18 65
08A8- 1A 85 1C 85 18 A5 1B 65
08B0- 15 85 1D 85 19 A0 00 B1
08B8- 12 91 1C C6 12 A5 12 C9
```

```
08C0- FF D0 02 C6 13 C6 1C A5
08C8- 1C C9 FF D0 02 C6 1D 20
08D0- 7F 08 D0 E1 A5 14 85 10
08D8- A5 15 85 11 A5 18 85 12
08E0- A5 19 85 13 60
*
```

7.210 HEXDUMP LISTEN VERSCHIEBEN

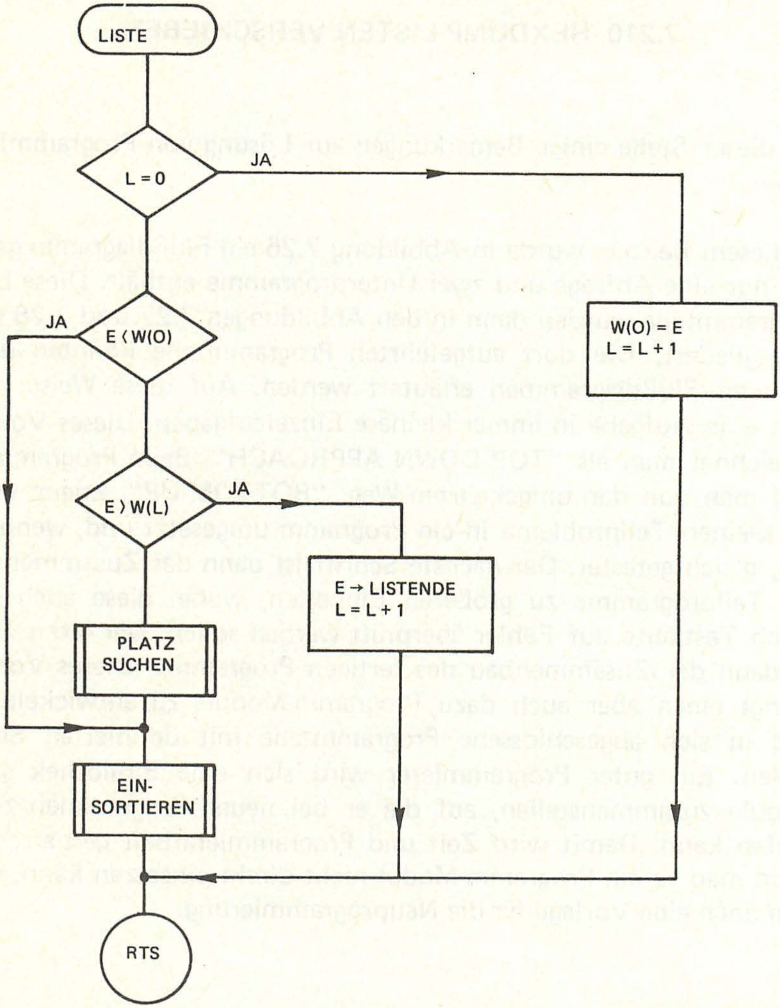
An dieser Stelle einige Bemerkungen zur Lösung von Programmieraufgaben.

In diesem Beispiel wurde in Abbildung 7.26 ein Flußdiagramm gezeigt, daß nur eine Abfrage und zwei Unterprogramme enthält. Diese beiden Programmteile wurden dann in den Abbildungen 7.27 und 7.28 weiter aufgegliedert. Die dort aufgeführten Programmteile könnten nun in weiteren Flußdiagrammen erläutert werden. Auf diese Weise zerlegt man eine Aufgabe in immer kleinere Einzelaufgaben. Dieses Vorgehen bezeichnet man als "TOP DOWN APPROACH". Beim Programmieren geht man nun den umgekehrten Weg: "BOTTOM-UP". Zuerst werden die kleinen Teilprobleme in ein Programm umgesetzt und, wenn möglich, gleich getestet. Der nächste Schritt ist dann das Zusammensetzen der Teilprogramme zu größeren Einheiten, wobei diese auch sofort durch Testläufe auf Fehler überprüft werden sollen. Der letzte Schritt ist dann der Zusammenbau des fertigen Programms. Dieses Vorgehen zwingt einen aber auch dazu, Programm-Module zu entwickeln. Dies sind in sich abgeschlossene Programmteile mit definierten Schnittstellen. Ein guter Programmierer wird sich eine Bibliothek solcher Module zusammenstellen, auf die er bei neuen Programmen zurückgreifen kann. Damit wird Zeit und Programmierarbeit gespart. Selbst wenn man so ein Programm-Modul nicht direkt einsetzen kann, so hat man doch eine Vorlage für die Neuprogrammierung.

7.24 Aufstellen einer sortierten Liste

Mit den folgenden Unterpogrammen kann eine Tabelle oder eine Liste erstellt werden, bei der die Elemente schon bei der Eingabe an die richtige Stelle einsortiert werden. Die Zahl der Elemente ist auf 256 beschränkt. Dadurch kann in den Programmteilen die indiziert indirekte Adressierung verwendet werden. Die Basisadresse der Tabelle

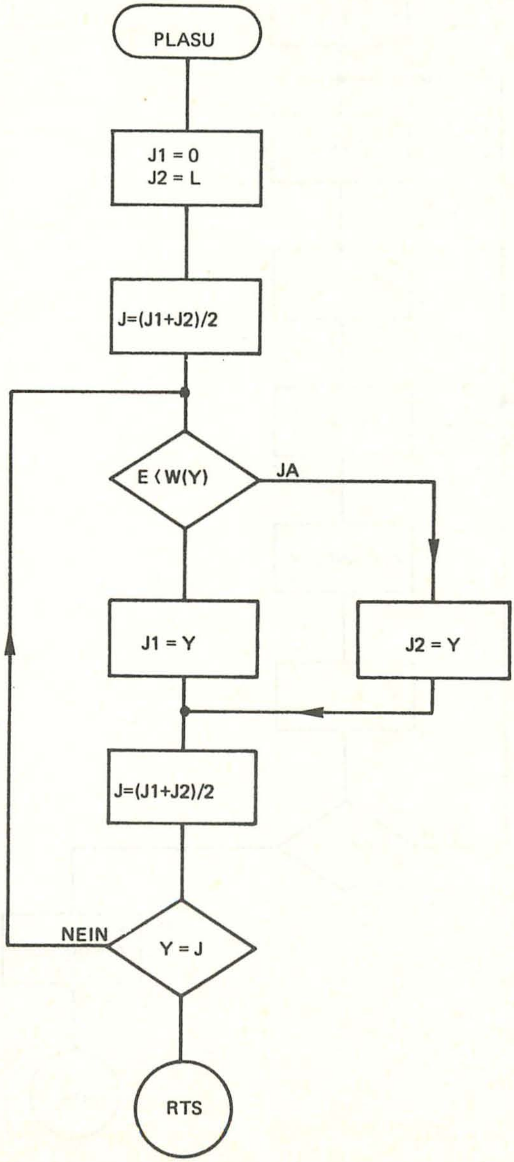
ist in den Zellen ANFL und ANFH gespeichert, Zelle L enthält die augenblickliche Länge der Tabelle, während die Zelle E den neu einzufügenden Wert enthält.



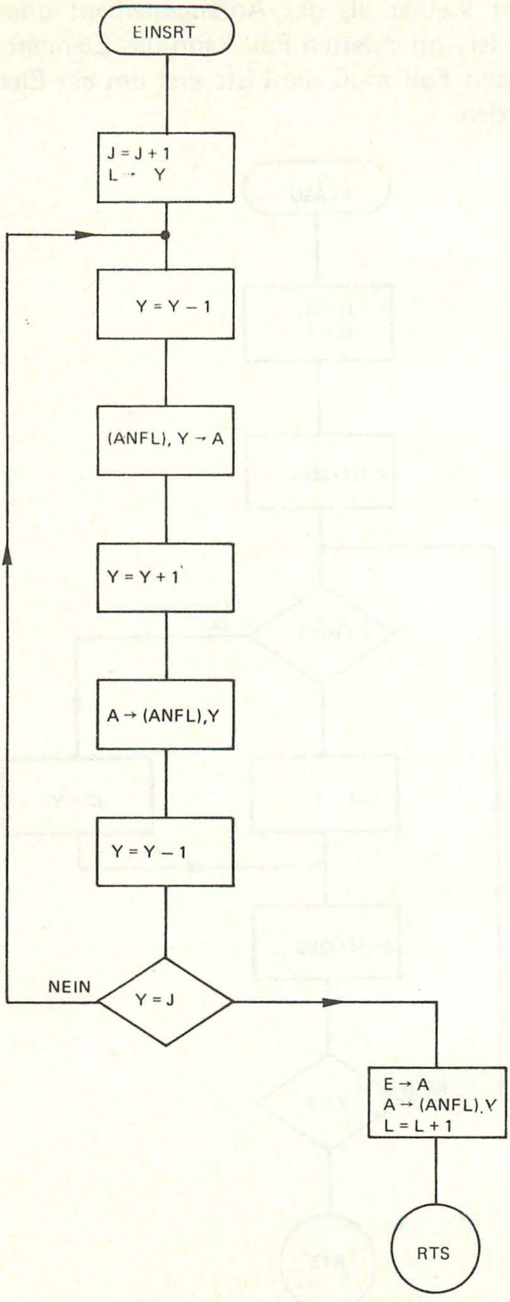
7.211 Flußdiagramm Aufstellen einer sortieren Liste

Das Flußdiagramm zeigt Abbildung 7.211. Das Programm beginnt mit der Abfrage, ob die Liste leer ist. Dann wird das Element einfach in die Liste geschrieben. Mit den beiden nächsten Abfragen wird festgestellt,

ob das Element kleiner als das Anfangselement oder größer als das letzte Element ist. Im zweiten Fall kann das Element gleich eingefügt werden, im ersten Fall muß die Liste erst um ein Element nach oben verschoben werden.



7.212 Flußdiagramm PLASU



7.213 Flußdiagramm EINSRT

Trifft dies alles nicht zu, so muß zuerst der richtige Platz für das Element gefunden werden (PLASU) und dann einsortiert werden. Dabei werden alle Elemente, die größer sind, um eine Stelle nach oben verschoben und somit Platz für das neue Element geschaffen. Die Flußdiagramme für PLASU und EINSRT zeigen die Abbildungen 7.212 und 7.213.

Im Programmteil PLASU wird ein binäres Suchverfahren angewendet. Die Liste wird halbiert und das neue Element mit dem Element in der Mitte der Liste verglichen. Je nachdem, ob es größer oder kleiner als dieses ist, wird auf die gleiche Weise, wiederum durch halbieren entweder in der unteren oder oberen Hälfte weitergesucht, solange bis keine Halbierungen mehr möglich sind. Damit ist der Platz des neuen Elementes gefunden. Dieses Verfahren wurde aus einem in BASIC geschriebenen Programmteil in Assembler übersetzt. Dieses Programm ist in Abbildung 7.214 aufgelistet und die entsprechenden Zeilen in das Programm 7.215 als Kommentar eingefügt.

```
1050 J1=0 : J2=L
1060 J = INT((J1+J2)/2)
1090 IF E$ < W$(J) THEN J2=J : GOTO 1110
1100 J1 = J
1110 IF J <> INT((J1+J2)/2) THEN 1060
1120 RETURN
```

7.214 BASIC-Programm PLASU

```
0800     ;
0800     ;
0800     ;******************************
0800     ;*                            *
0800     ;*  AUFSTELLEN EEINER SOR-    *
0800     ;*  TIERTEN LISTE             *
0800     ;*                            *
0800     ;******************************
0800     ;
0800     ;
0800     ANFL    EPZ $10
0800     ANFH    EPZ $11
0800     L       EPZ $12
0800     J       EPZ $14
0800     J1      EPZ $15
0800     J2      EPZ $16
0800     E       EPZ $17
```

```
0800                ;
0800                ;
0800                ;
0800 A512           LISTE  LDA L
0802 D00A                  BNE LIS1
0804 A000                  LDY #$00           ; LISTE LEER ?
0806 A517                  LDA E
0808 9110                  STA (ANFL),Y
080A E612                  INC L
080C D01F                  BNE LISRTS
080E                ;
080E A000           LIS1   LDY #$00
0810 A517                  LDA E              ; EINZUFUEGENDES ELEMENT
0812 D110                  CMP (ANFL),Y       ;KLEINER ALS ANFANGS-
0814 B009                  BCS LIS2           ;ELEMENT
0816 88                    DEY
0817 8414                  STY J
0819 205E08                JSR EINSRT
081C 18                    CLC
081D 900E                  BCC LISRTS
081F A412           LIS2   LDY L
0821 88                    DEY
0822 A517                  LDA E
0824 D110                  CMP (ANFL),Y       ;EINZUFUEGENDES ELEMENT
0826 9006                  BCC LIS3           ;GROESSER ALS LETZTES ELEMENT
0828 C8                    INY
0829 9110                  STA (ANFL),Y
082B E612                  INC L
082D 60             LISRTS RTS
082E 203408         LIS3   JSR PLASU
0831 4C5E08                JMP EINSRT
0834                ;
0834                ;
0834                ;******************************
0834                ;*                            *
0834                ;*  PLATZ SUCHEN              *
0834                ;*                            *
0834                ;******************************
0834                ;
0834                ;
0834 A512           PLASU  LDA L              ; J1=0:J2=L
0836 8516                  STA J2
0838 A900                  LDA #$00
083A 8515                  STA J1
083C 18                    CLC                ; J=INT((J1+J2)/2)
083D 6516                  ADC J2
083F 4A                    LSR
0840 8514                  STA J
0842 A414           S1     LDY J              ; IF E<W(J) THEN S2
0844 A517                  LDA E
0846 D110                  CMP (ANFL),Y
0848 9005                  BCC S2
084A 8415                  STY J1             ; ELSE J1=J
084C 18                    CLC
084D 9002                  BCC S3             ; GOTO S3
084F 8416           S2     STY J2             ; J2=J
0851 A515           S3     LDA J1             ; IF J <> INT((J1+J2)/2)
0853 18                    CLC
0854 6516                  ADC J2
0856 4A                    LSR
0857 8514                  STA J
0859 C414                  CPY J
085B D0E5                  BNE S1             ; THEN S1
085D 60                    RTS
```

```
085E              83  ;
085E              84  ;
085E              85  ;******************************
085E              86  ;*                            *
085E              87  ;*  EINSORTIEREN              *
085E              88  ;*                            *
085E              89  ;******************************
085E              90  ;
085E              91  ;
085E E614         92  EINSRT INC J
0860 A412         93         LDY L
0862 88           94  SRT1   DEY
0863 B110         95         LDA (ANFL),Y
0865 C8           96         INY
0866 9110         97         STA (ANFL),Y
0868 88           98         DEY
0869 C414         99         CPY J
086B DOF5        100         BNE SRT1
086D A517        101         LDA E
086F 9110        102         STA (ANFL),Y
0871 E612        103         INC L
0873 60          104         RTS
0874             105  ;
                 106         END
```

7.215 Programm LISTE

Alle Programme in diesem Teil sind nur für Listen mit 1 Byte langen Elementen aufgestellt worden. Durch entsprechende Unterprogramme lassen sich die Programme zur Handhabung von beliebig langen Listenelementen erweitern.

7.3 MENUE-Programme

Bei kurzen Maschinenprogrammen oder wenn diese Teil eines in einer höheren Programmiersprache geschriebenen Programms sind, wird man ohne MENUE-Technik auskommen. Diese ist aber unbedingt notwendig wenn vom Programm Eingaben über das Tastenfeld verlangt werden. In diesem Fall sollte man "interaktiv" programmieren. Das heißt, der Rechner gibt Anweisungen aus, welche Eingaben an dieser Stelle im Programm erwartet werden. Diese Anweisungen müssen erklärend sein, und nicht nur durch Ausgeben eines Zeichens erfolgen. Die Kosten für Speicherplatz sind so gering, daß man ruhig ein ganzes Wort oder einen Satz als Frage ausgeben kann. Für die in Kapitel 7.2 aufgestellten Programme wird als Beispiel ein Menue der folgenden Form angegeben:

T(ABELLE AUFSTELLEN
S(ORTIEREN

E(INFUEGEN
B(EENDEN
EINGABE:

Für dieses Programm benötigen wir ein Unterprogramm Textaus zur Ausgabe von Text auf den Bildschirm und einen Verteiler.
Für die Ausgabe von Text auf den Bildschirm gibt es zwei Möglichkeiten.

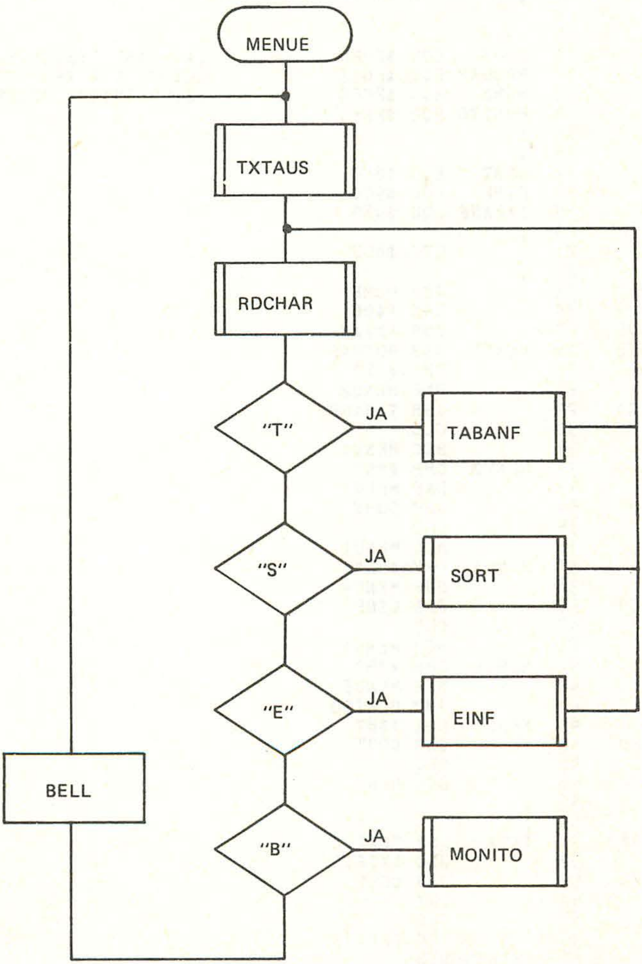
Der Text ist fortlaufend im Rechner gespeichert. Als letztes Zeichen des Textes ist ein Sonderzeichen vereinbart, das, wenn es vom Programm Textaus gelesen wird, zum Abbruch der Ausgabe führt. Die zweite Möglichkeit besteht darin, im ersten Byte des Textspeichers durch eine Zahl die Länge des Textes anzugeben, und die einzelnen Zeichen bei der Ausgabe mitzuzählen. Für unser Programm TXTAUS wollen wir die erste Methode wählen. Als Textendezeichen wird $00 gewählt (Abbildung 7.31).

```
L
TXTAUS LDA TEXT,X
       BEQ TXTRTS
       JSR COUT
       INX
       CLC
       BCC TXTAUS
TXTRTS RTS
;
TEXT   ASC "T(ABELLE AUFSTELLEN"
       HEX 8D
       ASC "S(ORTIEREN"
       HEX 8D
       ASC "E(INFUEGEN"
       HEX 8D
       ASC "B(EENDEN"
       HEX 8D8D
       ASC"EINGABE: "
       HEX 00
;
```

7.31 Unterprogramm TXTAUS

Das HEX-Zeichen 8D bedeutet Wagenrücklauf (RETURN) und gleichzeitig Zeilenvorschub. In dem eingegebenen Text kann das RETURN-Zeichen nicht eingegeben werden, da dieses Zeichen als Zeilenende vom Texteditor des Assemblers interpretiert wird.

Der Verteiler besteht in einer Kette von Abfragen (Abbildung 7.32). Auf alle Fälle sollte man sich gegen Fehlbedienungen schützen. Trifft keine der Abfragen zu, so wird das Zeichen $87 (BELL) ausgegeben und auf die Eingaberoutinen zurückgesprungen. Das vollständige Menueprogramm ist in Abbildung 7.32 aufgeführt. In diesem Programm wird ein Monitorprogramm HOME verwendet, das den Bildschirm löscht und den Cursor in die linke obere Ecke des Bildschirms setzt.



7.32 Flußdiagramm MENUE

7.31 Eingabe einer Dezimalzahl mit Umwandlung in Hexadezimalzahl

Bei der Eingabe in den Menueprogrammen wird oft die Eingabe eines Zahlenwertes verlangt. Diese Eingabe soll in dezimaler Form geschehen und vom Programm in die interne Bitdarstellung umgewandelt werden.

```
0800            ;
0800            ;
0800            ;****************************
0800            ;*                          *
0800            ;*      MENUPROGRAMM        *
0800            ;*                          *
0800            ;****************************
0800            ;
0800            ;
0800            COUT   EQU $FDED             ;AUSGABE EINES ASCII ZEICHENS
0800            RDCHAR EQU $FD35             ;EINGABE EINES ASCII ZEICHENS
0800            HOME   EQU $FC58             ;BILDSCHIRM LOESCHEN
0800            MONITO EQU $FF59
0800            ;
0800            ;
0800            SORT   EQU $880
0800            EINF   EQU $900
0800            TABANF EQU $980
0800            ;
0800                   ORG $800
0800            ;
0800 2058FC            JSR HOME
0803 A200              LDX #$00
0805 203808            JSR TXTAUS
0808 2035FD     MENU1  JSR RDCHAR
080B C9D4              CMP #"T"
080D D006              BNE MENU2
080F 208009            JSR TABANF
0812 18                CLC
0813 90F3              BCC MENU1
0815 C9D3       MENU2  CMP #"S"
0817 D006              BNE MENU3
0819 208008            JSR SORT
081C 18                CLC
081D 90E9              BCC MENU1
081F C9C5       MENU3  CMP #"E"
0821 D006              BNE MENU4
0823 200009            JSR EINF
0826 18                CLC
0827 90DF              BCC MENU1
0829 C9C2       MENU4  CMP #"B"
082B D003              BNE MENU5
082D 4C59FF            JMP MONITO
0830 A987       MENU5  LDA #$87
0832 20EDFD            JSR COUT
0835 18                CLC
0836 90D0              BCC MENU1
0838            ;
0838            ;
0838 BD4508     TXTAUS LDA TEXT,X
083B F007              BEQ TXTRTS
083D 20EDFD            JSR COUT
0840 E8                INX
0841 18                CLC
0842 90F4              BCC TXTAUS
0844 60         TXTRTS RTS
0845            ;
0845 D4A8C1     TEXT   ASC "T(ABELLE AUFSTELLEN"
0848 C2C5CC
```

```
084B CCC5A0
084E C1D5C6
0851 D3D4C5
0854 CCCCC5
0857 CE
0858 8D                    HEX 8D
0859 D3A8CF                ASC "S(ORTIEREN"
085C D2D4C9
085F C5D2C5
0862 CE
0863 8D                    HEX 8D
0864 C5A8C9                ASC "E(INFUEGEN"
0867 CEC6D5
086A C5C7C5
086D CE
086E 8D                    HEX 8D
086F C2A8C5                ASC "B(EENDEN"
0872 C5CEC4
0875 C5CE
0877 8D8D                  HEX 8D8D
0879 C5C9CE                ASC "EINGABE: "
087C C7C1C2
087F C5BAA0
0882 00                    HEX 00
0883                 ;
                           END
```

7.33 MENUEPROGRAMM

Dafür benötigen wir zwei Unterprogramme DIGIT und MULTEN. Im Programmteil DIGIT wird das eingegebene ASCII-Zeichen daraufhin untersucht, ob es eine Dezimalzahl darstellt. Trifft dies zu, so wird durch Ausblenden der vorderen vier Bit das ASCII-Zeichen in eine Binärzahl umgewandelt.

Die Zahl 1 wird durch ASCII $31 (APPLE = $B1) dargestellt. Durch Ausblenden mit der UND-Funktion und dem Bitmuster % 00001111 (= $0F) wird aus dem Zeichen $31 das Zeichen $01. Diese Zahl wird in der Hilfszelle INZ gespeichert und der Akkumulatorinhalt auf Null gesetzt. Ist das eingegebene Zeichen keine Dezimalzahl zwischen 0 und 9, so wird das Unterprogramm mit dem Zeichen im Akkumulator verlassen.

Die Eingabe geschieht dann in folgender Weise. Der Inhalt der beiden Zellen AL und AH wird zu Null gesetzt und die erste Zahl aus INZ dazuaddiert. Bevor nun die zweite Zahl zum Inhalt von AL und AH addiert wird, muß dieser mit dem Faktor 10 multipliziert werden. Dies geschieht im Programmteil MULTEN. Die Eingabeschleife wird verlassen wenn ein ASCII-Zeichen eingegeben wird, das keine Dezimalzahl darstellt.

Der Befehlsvorrat des 6502 kennt keinen Multiplikationsbefehl. Eine Multiplikation muß also durch Software ausgeführt werden. Die Multiplikation mit 10 lässt sich auf folgende Weise sehr leicht durchführen. Schiebt man den Inhalt einer Speicherzelle einmal nach links, so bedeutet dies eine Multiplikation mit 2 (Abbildung 7.34).

0	0	0	0	0	1	1	0	≙ 6
0	0	0	0	1	1	0	0	≙ 12

7.34 Multiplikation mit 2

Die Umkehrung davon, nämlich eine Stelle rechts schieben, bedeutet dann eine Division durch 2.

Der Speicherinhalt wird in einer Hilfszelle zwischengespeichert. Dann wird 2 mal linksgeschoben, dies entspricht einer Multiplikation mit 4. Der Inhalt der Hilfszelle wird dazuaddiert. Das entspricht einer Multiplikation mit 5. Die Multiplikation mit 10 wird dann durch ein nochmaliges linksschieben erreicht.

Im Programm MULTEN wird dieses Verfahren auf eine 16-bit Zahl angewendet. Das vollständige Programm zeigt Abbildung 7.35

```
0800           ;
0800           ;
0800           ;********************************
0800           ;*                              *
0800           ;* DEZIMALE ZAHLENEINGABE UND*
0800           ;* UMWANDLUNG IN BINAERZAHL    *
0800           ;*                              *
0800           ;********************************
0800           ;
0800           ;
0800           AL     EPZ $10
0800           AH     EPZ AL+1
0800           HL     EPZ AL+2
0800           HH     EPZ AL+3
0800           INZ    EPZ AL+4
0800           ;
0800           RDCHAR EQU $FD35
0800           COUT   EQU $FDED
0800           PRTBYT EQU $FDDA          ; AUSGABE EINES BYTES
0800           ;
0800                  ORG $800
0800 A900      ZEIN   LDA #$00
0802 8510             STA AL
0804 8511             STA AH
0806 201508    ZEIN1  JSR INCHAR
```

```
0809 D006                  BNE ZEIN2
080B 203208                JSR MULTEN
080E 18                    CLC
080F 90F5                  BCC ZEIN1
0811 205F08       ZEIN2    JSR OUT
0814 60                    RTS
0815              ;
0815              ;
0815 2035FD       INCHAR   JSR RDCHAR
0818 8514                  STA INZ
081A 20EDFD                JSR COUT
081D A514                  LDA INZ
081F 202308                JSR DIGIT
0822 60                    RTS
0823              ;
0823              ;
0823 C9B0         DIGIT    CMP #"0"
0825 900A                  BCC DIRTS
0827 C9C1                  CMP #"A"
0829 B006                  BCS DIRTS
082B 290F                  AND #%00001111
082D 8514                  STA INZ
082F A900                  LDA #$00
0831 60           DIRTS    RTS
0832              ;
0832              ;
0832 A510         MULTEN   LDA AL
0834 8512                  STA HL
0836 A511                  LDA AH
0838 8513                  STA HH
083A 0610                  ASL AL
083C 2611                  ROL AH              ; MAL 2
083E 0610                  ASL AL
0840 2611                  ROL AH              ; MAL 2
0842 18                    CLC
0843 A510                  LDA AL
0845 6512                  ADC HL
0847 8510                  STA AL
0849 A511                  LDA AH
084B 6513                  ADC HH
084D 8511                  STA AH              ; PLUS
084F 0610                  ASL AL
0851 2611                  ROL AH              ; MAL 2
0853 18                    CLC                 ; PLUS INZ
0854 A510                  LDA AL
0856 6514                  ADC INZ
0858 8510                  STA AL
085A 9002                  BCC MULRTS
085C E611                  INC AH
085E 60           MULRTS   RTS
085F              ;
085F              ;
085F A9A0         OUT      LDA #" "
0861 20EDFD                JSR COUT
0864 A511                  LDA AH
0866 20DAFD                JSR PRTBYT
0869 A510                  LDA AL
086B 20DAFD                JSR PRTBYT
086E 60                    RTS
086F              ;
086F              ;
                           END
```

7.35 Dezimale Zahleneingabe

7.32 Texteingabe in einen Textpuffer

Das folgende Programm TEXT ist ein Beispiel für Texteingabe. In den Rechner wird ein Text zeilenweise eingegeben. Jede Zeile kann, bevor sie in den Textspeicher übernommen wird, korrigiert werden. Dazu werden 5 Sonderzeichen vereinbart:

←	$08 ($88)	=	Cursor nach links ohne Löschen
→	$15 ($95)	=	Cursor nach rechts ohne Löschen
CTRL-A	$01 ($81)	=	Cursor nach links mit Löschen
CTRL-L	$0C ($8C)	=	Ein Zeichen löschen, bei der augenblicklichen Cursorposition
CTRL-E	$05 ($85)	=	Ein Zeichen einfügen, bei der augenblicklichen Cursorposition.

Bei Betätigen der Return-Taste wird der Text vom Zeilenanfang bis zur Curosrposition übertragen. Die Zeilenlänge des Textes soll (im Programmbeispiel) 40 Zeichen nicht überschreiten, eine Endeabfrage erfolgt aber nicht.

Rechner, die über einen Video- oder Fernsehausgang an einen Fernseher angeschlossen werden können, benutzen einen Teil des Speicherbereiches als Bildspeicher. Zeichen, die in diesen Bereich geschrieben werden, werden über einen Zeichengenerator in eine Impulsfolge für den Fernseher umgesetzt. Speichert man dort ein ASCII-Zeichen ab, so wird es auf dem Bildschirm ausgegeben.

Beim APPLE-II-Rechner liegt dieser Bereich bei den Adressen $400–$800. Eine Zeile dieses Bildspeichers wollen wir als Eingabezeile verwenden. Die Anfangsadresse ist $600, die Endadresse bei $623.

Das Programm ist in Abbildung 7.26 gezeigt. Benötigt werden vom Hauptprogramm lediglich die Eingaberoutine RDCHAR, die Festlegung der Bildspeicheradresse und die Festlegung des Textpuffers TEXTBU. Das Unterprogramm HOME (Bildschirm löschen) kann durch ein Maschinenprogramm ersetzt werden, das den Bildspeicher mit Blanks (ASCII $20, APPLE $A0) vollschreibt.

In den Zeilen 28 bis 73 ist ein Verteiler ähnlich dem im Menueprogramm aufgebaut. Der Rücksprung zu Adresse TEX1 oder TEX0 erfolgt

mit dem Befehl BCC <MARKE> . Um an dieser Stelle etwas Platz zu sparen wird vorausgesetzt, daß im Unterprogramm vor dem RTS ein CLC ausgeführt wird.

Zusätzlich zu den schon angegebenen Abfragen werden noch zwei weitere eingeführt. Mit RETURN wird der Text bis zur augenblicklichen Cursorposition in einen Textspeicher geschrieben und die Zeile gelöscht, mit CTRL-C wird das Programm abgebrochen.

In den Unterprogrammen werden zwei Zeiger benutzt. Der Zeiger L zeigt auf die augenblickliche Cursorstellung, der Zeiger Z auf das Ende der Textzeile.

Bei Eingabe eines Zeichens wird dieses auf den Bildschirm ausgegeben und der Cursor (–) eine Stelle rechts geschoben.

Im Unterprogramm LINKS wird bei Eingabe des Codes $88 der augenblickliche Cursor eine Stelle nach links geschoben. Um dies sichtbar zu machen wird das Zeichen invers dargestellt. Dies geschieht beim APPLE durch Nullsetzen der beiden höchsten Bit (AND mit % 0 0 1 1 1 1 1 1). Umgekehrt wird im Unterprogramm RECHTS durch das Zeichen $95 der laufende Cursor eine Stelle nach rechts geschoben. Dabei wird über die ODER-Funktion (ORA mit % 1 1 0 0 0 0 0 0) die beiden höchsten Bit wieder auf Eins gesetzt und das Zeichen wieder normal ausgegeben.

Bei Eingabe des Codes $8C (CTRL-A) wird das letzte Zeichen gelöscht und der Endzeiger eine Stelle nach links geschoben (Programmteil LOE1). Das Löschen eines Zeichens innerhalb des Textes geschieht im Programmteil LOE2. Durch Eingabe von CTRL-L ($8C) wird das Zeichen bei der augenblicklichen Cursorposition gelöscht und der rechts davon stehende Text eine Stelle nach links verschoben.

Schließlich wird im Programmteil EINF durch das Zeichen $85 (CTRL-E) der augenblickliche Platz zum Einfügen von Zeichen geschaffen.

In diesem Unterprogramm bedarf die Befehlsfolge in den Zeilen 141 bis 146 noch einer Erläuterung.

Beim Verschieben des Textes wird in Zeile 142 verglichen, ob der augenblickliche Cursorstand schon erreicht ist. Ist dies nicht der Fall so muß vor dem Rücksprung das Y-Register noch einmal um Eins erniedrigt werden. Damit wird aber der Inhalt des Statusregisters verändert und somit die Bedingung für die Verzweigung BEQ EINF1 in Zeile 146. Abhilfe wird dadurch geschaffen, daß der Inhalt des Statusregisters vor dem DEY auf den Stapelspeicher geschrieben wird und nachher vor der Sprungbedingung wieder von dort geholt wird. Das letzte Unterprogramm SPCHRN schreibt den Text von Anfang der Zeile bis zur augenblicklichen Cursorposition.

Die Erweiterung dieses Zeileneditors zu einem zeilenorientierten Texteditor sei dem Leser als Programmierübung überlassen. Die Zeilen müssen durch Zeilennummern gekennzeichnet und im Textspeicher abgelegt werden. Durch Befehle wie LIST oder NLIST kann dieser Textspeicher auf den Bildschirm gebracht werden, wobei mit NLIST z. B. keine Zeilennummern ausgegeben werden. Für das Einfügen und Löschen von Zeilen können die Programme in 7.2 als Beispiele dienen.

```
0800            2  ;
0800            3  ;
0800            4  ;*****************************
0800            5  ;*                           *
0800            6  ;* TEXTVERARBEITUNG          *
0800            7  ;*                           *
0800            8  ;*****************************
0800            9  ;
0800           10  ;
0800           11  ANFL   EPZ $10
0800           12  L      EPZ $12
0800           13  E      EPZ $13
0800           14  Z      EPZ $14
0800           15  ;
0800           16  ;
0800           17  HOME   EQU $FC58            ;BILDSCHIRM LOESCHEN
0800           18  MONITO EQU $FF59            ;RUECKSPRUNG MONITOR
0800           19  CRLF   EQU $FD8E            ;AUSGABE VON CRLF
0800           20  RDCHAR EQU $FD35            ;ASCII ZEICHEN LESEN
0800           21  BSPL   EQU $00
0800           22  BSPH   EQU $06              ;ADRESSE BILDZEILE
0800           23  TEXTBU EQU $1000
0800           24  ;
0800           25  ;
0800           26  ;
0800           27  ;
0800 2058FC    28  TEXT   JSR HOME
0803 A000      29         LDY #$00
0805 8412      30         STY L
0807 8414      31         STY Z
0809 8410      32         STY ANFL
080B A906      33         LDA #BSPH
080D 8511      34         STA ANFL+1
080F A9DF      35  TEXO   LDA #$DF
0811 9110      36         STA (ANFL),Y
0813 2035FD    37  TEX1   JSR RDCHAR
0816 8513      38         STA E
0818 C983      39         CMP #$83             ;END OF TEXT
081A D003      40         BNE TEX2
081C 4C59FF    41         JMP MONITO
081F C988      42  TEX2   CMP #$88             ;CURSOR LINKS
0821 D005      43         BNE TEX3
0823 206508    44         JSR LINKS
0826 90EB      45         BCC TEX1
0828 C995      46  TEX3   CMP #$95             ;CURSOR RECHTS
082A D005      47         BNE TEX4
082C 207208    48         JSR RECHTS
082F 90E2      49         BCC TEX1
0831 C981      50  TEX4   CMP #$81             ;CTRL-A
0833 D005      51         BNE TEX5
0835 208308    52         JSR LOE1
0838 90D9      53         BCC TEX1
083A C98C      54  TEX5   CMP #$8C             ;CTRL-L
083C D005      55         BNE TEX6
083E 209608    56         JSR LOE2
0841 90D0      57         BCC TEX1
0843 C985      58  TEX6   CMP #$85             ;CTRL-E
0845 D005      59         BNE TEX7
0847 20BB08    60         JSR EINF
```

```
084A 90C7               BCC TEX1
084C C98D        TEX7   CMP #$8D              ;RETURN
084E D005               BNE TEX8
0850 20D908             JSR SPCHRN
0853 90AB               BCC TEXT
0855 A412        TEX8   LDY L                 ;EINFUEGEN DES ZEICHENS
0857 9110               STA (ANFL),Y
0859 C8                 INY
085A 8412               STY L
085C C414               CPY Z
085E 90B3               BCC TEX1              ;IM TEXT
0860 E614               INC Z
0862 B0AB               BCS TEX0              ;AM ENDE
0864             ;
0864             ;
0864 00          ENDE   HEX 00
0865 A412        LINKS  LDY L
0867 88                 DEY
0868 B110               LDA (ANFL),Y
086A 293F               AND #%00111111        ;BIT8 UND BIT7 = 0
086C 9110               STA (ANFL),Y
086E 8412               STY L
0870 18                 CLC
0871 60                 RTS
0872             ;
0872 A412        RECHTS LDY L
0874 C414               CPY Z
0876 F009               BEQ RERTS
0878 B110               LDA (ANFL),Y
087A 09C0               ORA #%11000000        ;BIT8 UND BIT7 = 1
087C 9110               STA (ANFL),Y
087E C8                 INY
087F 8412               STY L
0881 18          RERTS  CLC
0882 60                 RTS
0883             ;
0883 A412        LOE1   LDY L                 ;LINKS SCHIEBEN MIT LOESCHEN
0885 B110               LDA (ANFL),Y
0887 AA                 TAX
0888 A9A0               LDA #$A0
088A 9110               STA (ANFL),Y
088C 8A                 TXA
088D 88                 DEY
088E 9110               STA (ANFL),Y
0890 8412               STY L
0892 8414               STY Z
0894 18                 CLC
0895 60                 RTS
0896             ;
0896 A412        LOE2   LDY L                 ;ZEICHEN BEI CURSOR
0898 C414               CPY Z
089A F01D               BEQ LLRTS
089C C8          LO     INY                   ;POSITION LOESCHEN
089D C414               CPY Z
089F F009               BEQ LL
08A1 B110               LDA (ANFL),Y
08A3 88                 DEY
08A4 9110               STA (ANFL),Y
08A6 C8                 INY
08A7 18                 CLC
08A8 90F2               BCC LO
08AA A414        LL     LDY Z
08AC B110               LDA (ANFL),Y
08AE AA                 TAX
08AF A9A0               LDA #" "
08B1 9110               STA (ANFL),Y
08B3 8A                 TXA
08B4 88                 DEY
08B5 9110               STA (ANFL),Y
```

```
08B7 8414                 STY Z
08B9 18          LLRTS    CLC
08BA 60                   RTS
08BB             ;
08BB A412        EINF     LDY L                  ; ZEICHEN AN CURSOR
08BD C414                 CPY Z                  ;POSITION EINFUEGEN
08BF F016                 BEQ EINRTS
08C1 A414                 LDY Z
08C3 B110        EINF1    LDA (ANFL),Y
08C5 C8                   INY
08C6 9110                 STA (ANFL),Y
08C8 88                   DEY
08C9 C412                 CPY L
08CB 08                   PHP
08CC 88                   DEY
08CD 28                   PLP
08CE D0F3                 BNE EINF1
08D0 C8                   INY
08D1 A9A0                 LDA #" "
08D3 9110                 STA (ANFL),Y
08D5 E614                 INC Z
08D7 18          EINRTS   CLC
08D8 60                   RTS
08D9             ;
08D9 A000        SPCHRN   LDY #$00
08DB B110        SP1      LDA (ANFL),Y
08DD 990010               STA TEXTBU,Y
08E0 C8                   INY
08E1 C412                 CPY L
08E3 D0F6                 BNE SP1
08E5 A039                 LDY #$39
08E7 A9A0                 LDA #" "
08E9 9110        SP2      STA (ANFL),Y
08EB 88                   DEY
08EC D0FB                 BNE SP2
08EE 18                   CLC
08EF 60                   RTS
08F0             ;
                 FINI     END

***** END OF ASSEMBLY


          **************************
          *                        *
          * SYMBOL TABLE -- V 1.5  *
          *                        *
          **************************

LABEL. LOC.  LABEL. LOC.  LABEL. LOC.

** ZERO PAGE VARIABLES:

ANFL   0010  L      0012  E      0013  Z      0014

** ABSOLUTE VARABLES/LABELS

HOME   FC58  MONITO FF59
CRLF   FD8E  RDCHAR FD35  BSPL   0000  BSPH   0006  TEXTBU 1000  TEXT   0800
TEX0   080F  TEX1   0813  TEX2   081F  TEX3   0828  TEX4   0831  TEX5   083A
TEX6   0843  TEX7   084C  TEX8   0855  ENDE   0864  LINKS  0865  RECHTS 0872
RERTS  0881  LOE1   0883  LOE2   0896  LO     089C  LL     08AA  LLRTS  08B9
EINF   08BB  EINF1  08C3  EINRTS 08D7  SPCHRN 08D9  SP1    08DB  SP2    08E9
FINI   08F0

SYMBOL TABLE STARTING ADDRESS:6000
SYMBOL TABLE LENGTH:013A
```

7.36 Programm Textverarbeitung

```
0800- 20 58 FC A0 00 84 12 84
0808- 14 84 10 A9 06 85 11 A9
0810- DF 91 10 20 35 FD 85 13
0818- C9 83 D0 03 4C 59 FF C9
0820- 88 D0 05 20 65 08 90 EB
0828- C9 95 D0 05 20 72 08 90
0830- E2 C9 81 D0 05 20 83 08
0838- 90 D9 C9 8C D0 05 20 96
0840- 08 90 D0 C9 85 D0 05 20
0848- BB 08 90 C7 C9 8D D0 05
0850- 20 D9 08 90 AB A4 12 91
0858- 10 C8 84 12 C4 14 90 B3
0860- E6 14 B0 AB 00 A4 12 88
0868- B1 10 29 3F 91 10 84 12
0870- 18 60 A4 12 C4 14 F0 09
0878- B1 10 09 C0 91 10 C8 84
0880- 12 18 60 A4 12 B1 10 AA
0888- A9 A0 91 10 8A 88 91 10
0890- 84 12 84 14 18 60 A4 12
0898- C4 14 F0 1D C8 C4 14 F0
08A0- 09 B1 10 88 91 10 C8 18
08A8- 90 F2 A4 14 B1 10 AA A9
08B0- A0 91 10 8A 88 91 10 84
08B8- 14 18 60 A4 12 C4 14 F0
08C0- 16 A4 14 B1 10 C8 91 10
08C8- 88 C4 12 08 88 28 D0 F3
08D0- C8 A9 A0 91 10 E6 14 18
08D8- 60 A0 00 B1 10 99 00 10
08E0- C8 C4 12 D0 F6 A0 39 A9
08E8- A0 91 10 88 D0 FB 18 60
08F0- 73
*
```

7.37 HEXDUMP Programm Textverarbeitung

7.4 Mathematikroutinen

Hier wollen wir je ein Beispiel für Multiplikation und Division betrachten.

7.41 8 x 8-bit Multiplikation

Bei der Multiplikation sollen 2 8-bit Binärzahlen, ohne Vorzeichen, miteinander zu einem 16-bit Produkt multipliziert werden.

Als Algorithmus wird die "Schiebe und Addiere" -Methode verwendet, die man sich am Besten an einem Beispiel klarmacht. Die beiden Zahlen 6 und 2 sollen miteinander multipliziert werden.

Multiplikant = 6	0	0	0	0	0	1	1	0								
Multiplikator = 2	0	0	0	0	0	0	1	0								
Ergebnis:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ERG ERG + 1

Der Multiplikator (oder der Multiplikant) wird eine Stelle rechts geschoben. Ist das Carry-Bit 0 so wird das Ergebnis einmal rechts geschoben. Ist das Carry-Bit 1, so wird der Multiplikant (oder der Multiplikator) erst zu ERG addiert und dann eine Stelle rechts geschoben.

In unserem Beispiel ist nach dem 2. Schieben des Multiplikators der Inhalt von ERG und ERG + 1:

0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

ERG ERG + 1

Nachdem im Multiplikator keine Eins mehr enthalten ist, wird das Ergebnis noch 6 mal nach rechts geschoben und wir erhalten das Ergebnis:

0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	= 12

ERG ERG + 1

Das Programm in Abbildung 7.41 ist die 8 x 8-bit Multiplikationsroutine.

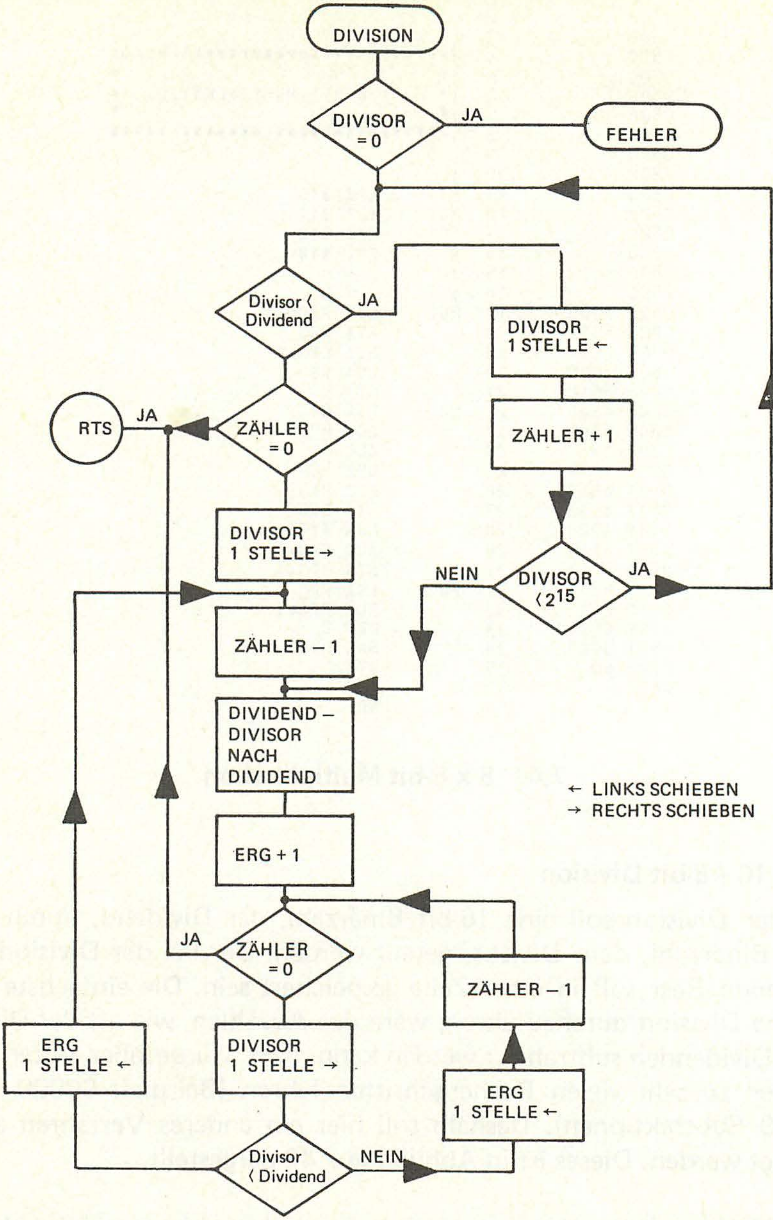
```
0800            ;
0800            ;
0800            ;*****************************
0800            ;*                           *
0800            ;*  8 X 8 BIT MULTIPLKATION  *
0800            ;*                           *
0800            ;*****************************
0800            ;
0800            ;
0800            M1      EPZ $10
0800            M2      EPZ $11
0800            ERG     EPZ $12
0800            Z       EPZ $14
0800            ;
0800            ;
0800 A900       MUL     LDA #$00
0802 8512               STA ERG
0804 8513               STA ERG+1
0806 A908               LDA #$08
0808 8514               STA Z
080A 4610       MUL1    LSR M1
080C 900D               BCC MUL2
080E 18                 CLC
080F A511               LDA M2
0811 6512               ADC ERG
0813 8512               STA ERG
0815 A900               LDA #$00
0817 6513               ADC ERG+1
0819 8513               STA ERG+1
081B 4612       MUL2    LSR ERG
081D 6613               ROR ERG+1
081F C614               DEC Z
0821 D0E7               BNE MUL1
0823 60                 RTS
0824            ;
                        END
```

7.41 8 x 8-bit Multiplikation

7.42 16 / 8-bit Division

Bei der Division soll eine 16-bit Binärzahl, der Dividend, durch eine 8-bit Binärzahl, dem Divisor, geteilt werden. Der bei der Division verbleibende Rest soll in einer Zelle gespeichert sein. Die einfachste Art, so eine Division durchzuführen, wäre das Abzählen, wie oft der Divisor vom Dividenden subtrahiert werden kann. Das würde aber unter Umständen zu sehr vielen Rechenschritten führen (Beispiel: 60000 : 2 = 30000 Subtraktionen). Deshalb soll hier ein anderes Verfahren angewendet werden. Dieses ist in Abbildung 7.42 dargestellt.

Der Divisor wird so lange nach links geschoben, bis er größer als der Dividend ist. Dies wird in einem Zähler mitgezählt. Danach wird einmal nach rechts geschoben und der Zähler um Eins erniedrigt. Dann beginnt die Division. Der Divisor ist jetzt kleiner, höchstens gleich dem Dividenden und wird von diesem subtrahiert, und im Ergebnis Eins addiert.



7.42 Flußdiagramm 16 / 8 BIT DIVISION

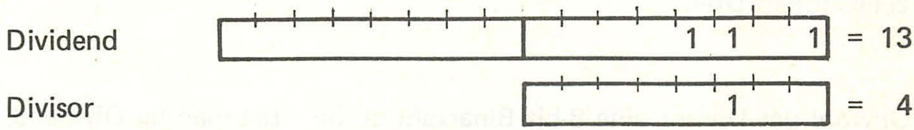
Dann wird der Divisor einmal nach rechts geschoben und der Zähler wieder um Eins erniedrigt. Ist dieser Null, dann ist die Division beendet. Wenn nicht, wird verglichen ob der Divisor kleiner als der Dividend ist. Wenn dies zutrifft, wird das Ergebnis 1 Stelle links geschoben, vom Dividend der Divisor abgezogen und zum Ergebnis Eins addiert.

Dies wird solange durchgeführt bis der Divisor größer als der Dividend ist. Ist der Inhalt des Zählers dabei noch nicht Null, muß das Ergebnis noch solange nach rechts geschoben werden, bis dieser Null ist, erst dann ist die Division beendet.

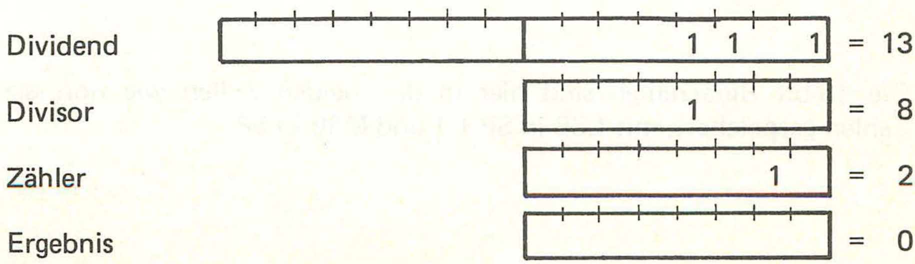
Im Flußdiagramm sind noch 3 weitere Abfragen. Ist der Divisor Null, so kann keine Division erfolgen. Ist gleich zu Beginn der Dividend kleiner als der Divisor, so braucht die Division ebenfalls nicht durchgeführt werden. Der Dividend enthält dann den verbleibenden Rest.

Die dritte Abfrage bezieht sich auf das Linksschieben des Divisors. Dieses darf nur solange durchgeführt werden, solange der Divisor kleiner, höchstens gleich 2^{15} ist. Dann muß auf alle Fälle eine Subtraktion erfolgen.

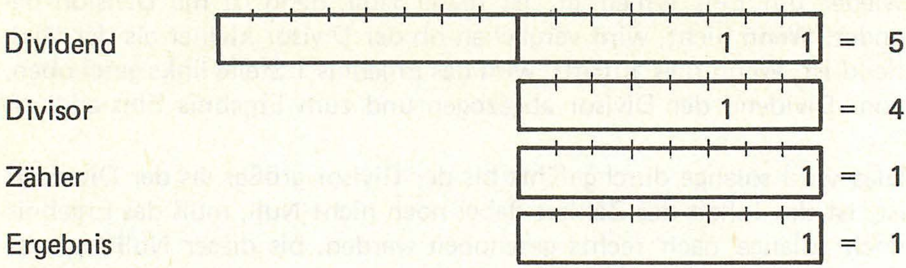
Dies alles sei an einem Beispiel 13 : 4 = 3 Rest 1 gezeigt.



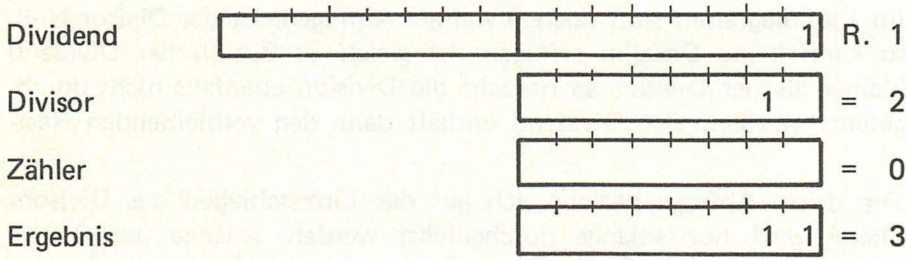
Nach 3 – 1 = 2 mal Linksschieben des Divisors



Nach der ersten Subtraktion und dem Vergleich



Nach der zweiten Subtaktion und dem Vergleich



Da der Dividend kleiner ist als der Divisor und der Zähler Null ist, ist die Divison beendet. Im Dividend verbleibt Eins als Rest, die Ergebniszelle enthält Drei.

Obwohl der Divisor eine 8-bit Binärzahl ist, benötigt man für Dividend, Divisor und Ergebnis 2 Doppelzellen. Im Programm DIV (Abbildung 7.43) wird das Y-Register als Zähler verwendet.

Die 16-bit Binärzahlen sind hier in den beiden Zellen wie normale Zahlen gespeichert, mit LSB in SP + 1 und MSB in SP.

```
0800            ;
0800            ;
0800            ;*******************************
0800            ;*                             *
0800            ;* 16/8 BIT DIVISION           *
0800            ;*                             *
0800            ;*******************************
0800            ;
0800            ;
0800            DIVDND EPZ $10
0800            DIVISO EPZ $12
0800            ERG    EPZ $14
0800            ;
0800            ; .
0800 A512       DIV    LDA DIVISO
0802 D004              BNE DIV1
0804 A513              LDA DIVISO+1
0806 F062              BEQ DIVNUL           ; DIVISION DURCH NULL
0808 A000       DIV1   LDY #$00
080A 8414              STY ERG
080C 8415              STY ERG+1
080E 205608     DIV2   JSR COMPAR           ; DIVISOR <DIVIDENT
0811 B009              BCS WEITER
0813 C8                INY
0814 0613              ASL DIVISO+1
0816 2612              ROL DIVISO
0818 300C              BMI WEIT1            ; DIVISOR> 2^15
081A 10F2              BPL DIV2
081C EA         WEITER NOP
081D C000              CPY #$00
081F F034              BEQ DIVRTS           ; KEINE DIVISION
0821 4612              LSR DIVISO           ; DIVISOR 1 STELLE -->
0823 6613              ROR DIVISO+1
0825 88         WEIT2  DEY
0826 38         WEIT1  SEC
0827 A511              LDA DIVDND+1         ; SUBTRAKTION
0829 E513              SBC DIVISO+1
082B 8511              STA DIVDND+1
082D A510              LDA DIVDND
082F E512              SBC DIVISO
0831 8510              STA DIVDND
0833 E615              INC ERG+1            ; ERGEBNIS+1
0835 D002              BNE DIV3
0837 E614              INC ERG
0839 EA         DIV3   NOP
083A C000              CPY #$00             ; ZAEHLER=0:FERTIG
083C F017              BEQ DIVRTS
083E 4612              LSR DIVISO
0840 6613              ROR DIVISO+1         ; DIVISOR 1 STELLE -->
0842 205608            JSR COMPAR           ; DIVISOR <DIVIDEND
0845 B007              BCS DIV4             ; JA
0847 0615              ASL ERG+1            ; ERGEBNIS 1 STELLE <--
0849 2614              ROL ERG
084B 18                CLC
084C 90D7              BCC WEIT2
084E 0615       DIV4   ASL ERG+1            ; NEIN
0850 2614              ROL ERG
0852 88                DEY
0853 D0E4              BNE DIV3
0855 60         DIVRTS RTS
0856            ;
0856            ;
0856 A512       COMPAR LDA DIVISO           ; DIVISOR<DIVIDEND C=0
0858 C510              CMP DIVDND           ; DIVISOR=DIVIDEND C=0
085A 900D              BCC CRTS             ; DIVISOR > DIVIDENT C=1
085C F002              BEQ C2
085E B009              BCS CRTS
0860 A513       C2     LDA DIVISO+1
0862 C511              CMP DIVDND+1
```

```
0864 F002       71          BEQ C1
0866 B001       72          BCS CRTS
0868 18         73   C1     CLC
0869 60         74   CRTS   RTS
086A            75   ;
086A            76   ;
086A 00         77   DIVNUL BRK
086B            78   ;
                79          END
```

```
0800- A5 12 D0 04 A5 13 F0 62
0808- A0 00 84 14 84 15 20 56
0810- 08 B0 09 C8 06 13 26 12
0818- 30 0C 10 F2 EA C0 00 F0
0820- 34 46 12 66 13 88 38 A5
0828- 11 E5 13 85 11 A5 10 E5
0830- 12 85 10 E6 15 D0 02 E6
0838- 14 EA C0 00 F0 17 46 12
0840- 66 13 20 56 08 B0 07 06
0848- 15 26 14 18 90 D7 06 15
0850- 26 14 88 D0 E4 60 A5 12
0858- C5 10 90 0D F0 02 B0 09
0860- A5 13 C5 11 F0 02 B0 01
0868- 18 60 00
*
```

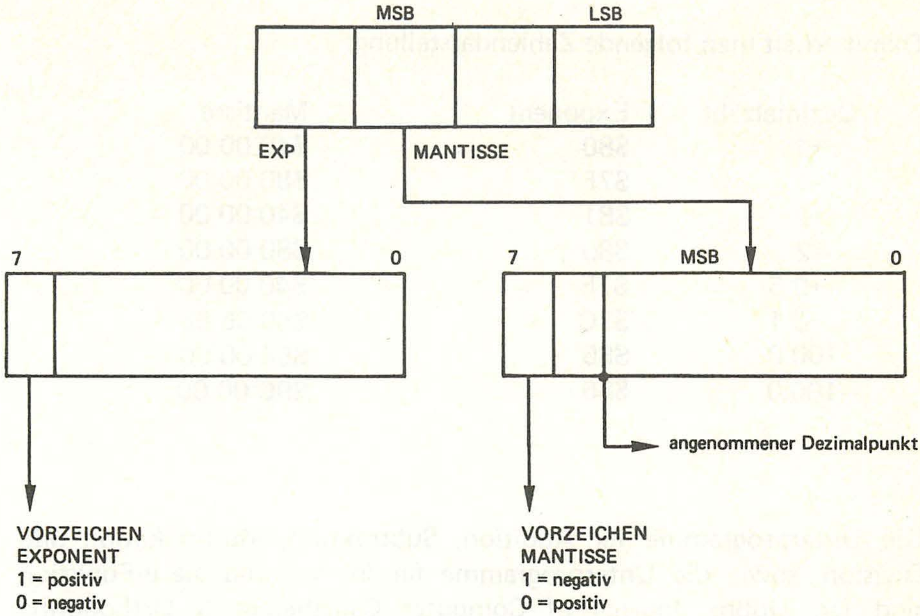
7.43 16 / 8 BIT DIVISION

7.5 Gleitkommaarithmetik

Für Berechnungen, die über die Zahlengröße von 2^{16} hinausgehen, wird die Darstellung einer Zahl in Gleitkommaformat verwendet.

Die Zahl wird aufgeteilt in Exponent und Mantisse. Für den Exponenten wird ein Byte reserviert, für die Mantisse entweder 10 Byte für 10 Dezimalstellen (ATARI) oder 3 Byte für Binärdarstellung (APPLE, AIM . . .).

Für die folgenden Programme wird die letztere Darstellung gewählt.



7.51 Gleitkommadarstellung

Das Vorzeichen des Exponenten ist Bit 7. Für positive Zahlen ist es 1, für negative Zahlen ist es 0. Damit erhalten wir folgende Zahlendarstellung:

%1 0 0 0 0 0 0 1 = \$81 = 1
%1 0 0 0 0 0 0 0 = \$80 = 0
%0 1 1 1 1 1 1 1 = \$7F = –1

Die Mantisse wird im 2-er Complement dargestellt.

Das Vorzeichen ist auch hier Bit 7 von MSB. Für negative Mantissen ist es 1, für positive Mantissen ist es 0. Die Mantisse wird, zur Erhaltung der Rechengenauigkeit immer so normalisiert, daß die beiden höchsten Bit von MSB verschieden sind (Ausnahme: Zahlen kleiner 2^{-128}).

Damit erhält man folgende Zahlendarstellung:

Dezimalzahl	Exponent	Mantisse
+1	$80	$40 00 00
−1	$7F	$80 00 00
+2	$81	$40 00 00
−2	$80	$80 00 00
+0.5	$7F	$40 00 00
+0.1	$7C	$66 66 66
100.0	$86	$64 00 00
−100.0	$86	$9C 00 00

Die Unterprogramme für Addition, Subtraktion, Multiplikation und Division, sowie die Unterprogramme für ln, log und die e-Funktion sind Dr. Dobbs Journal of Computer Calisthenics & Orthodontia (Volume 1, mit Fehlerberichtigung) entnommen.

Die Unterprogramme belegen folgende Zellen in der Zero-Page

Akkumulator 1	Exponent: $08, Mantisse: $09, $0A, $0B
Akkumulator 2	Exponent: $04, Mantisse: $05, $06, $07

Als Hilfszellen sind alle Speicherplätze von $02 bis $10 belegt.

1. Unterprogramm FADD
   Anfangsadresse $849
   Z1 in Akkumulator 1, Z2 in Akkumulator 2
   Ergebnis Z1 + Z2 in Akkumulator 1.

2. Unterprogramm FSUB
   Anfangsadresse $843
   Z1 in Akkumulator 1, Z2 in Akkumulator 2
   Ergebnis Z2 − Z1 in Akkumulator 1.

3. Unterprogramm FMUL
   Anfangsadresse $867
   Z1 in Akkumulator 1, Z2 in Akkumulator 2
   Ergebnis Z1 * Z2 in Akkumulator 1.

4. Unterprogramm FDIV
   Anfangsadresse $88D
   Z1 in Akkumulator 1, Z2 in Akkumulator 2
   Ergebnis Z2/Z1 in Akkumulator 1.

5. Unterprogramm SWAP
   Anfangsadresse $81C
   Austausch Akkumulator 1 mit Akkumulator 2.

6. Unterprogramm FLOAT
   Anfangsadresse $82C
   Umwandlung einer 16-bit Binärzahl mit MSB in $09 und LSB in $09 in eine Gleitkommazahl.
   Ergebnis in Akkumulator 1.

7. Unterprogramm FIX
   Anfangsadresse $8D5
   Umwandlung einer Gleitkommazahl kleiner 65 536 in Akkumulator 1 in eine 16-bit Festkommazahl mit MSB in $09 und LSB in $0A.

8. Unterprogramm LOG
   Anfangsadresse $8F6
   Bildung des natürlichen Logarithmus einer Zahl Z in Akkumulator 1,
   Ergebnis in Akkumulator 1.

9. Unterprogramm LOG10
   Anfangsadresse $9B5
   Bildung des dekadischen Logarithmus einer Zahl Z in Akkumulator 1.
   Ergebnis in Akkumulator 1.

10. Unterprogramm EXPO
    Anfangsadresse $9E2
    Berechnung von e^Z mit Z und anschließendem Ergebnis in Akkumulator 1.

Die Programme und ihre Speicherauszüge sind in den Abbildungen 7.52 bis 7.55 gezeigt.

```
0800            ;
0800            ;
0800            ;******************************
0800            ;*                            *
0800            ;* FLOATING POINT ROUTINEN    *
0800            ;*                            *
0800            ;******************************
0800            ;
0800            SIGN   EPZ $03
0800            X2     EPZ $04
0800            M2     EPZ $05
0800            X1     EPZ $08
0800            M1     EPZ $09
0800            E      EPZ $0C
0800            Z      EPZ $10
0800            T      EPZ $14
0800            SEXP   EPZ $18
0800            INT    EPZ $1C
0800            ;
0800            ;
0800 18         ADD    CLC
0801 A202              LDX #$02
0803 B509       ADD1   LDA M1,X
0805 7505              ADC M2,X
0807 9509              STA M1,X
0809 CA                DEX
080A 10F7              BPL ADD1
080C 60                RTS
080D 0603       MD1    ASL SIGN
080F 201208            JSR ABSWAP
0812 2409       ABSWAP BIT M1
0814 1005              BPL ABSWA1
0816 207F08            JSR FCOMPL
0819 E603              INC SIGN
081B 38         ABSWA1 SEC
081C A204       SWAP   LDX #$04
081E 940B       SWAP1  STY E-1,X
0820 B507              LDA X1-1,X
0822 B403              LDY X2-1,X
0824 9407              STY X1-1,X
0826 9503              STA X2-1,X
0828 CA                DEX
0829 D0F3              BNE SWAP1
082B 60                RTS
082C A98E       FLOAT  LDA #$8E
082E 8508              STA X1
0830 A509       NORM1  LDA M1
0832 C9C0              CMP #$C0
0834 300C              BMI RTS1
0836 C608              DEC X1
0838 060B              ASL M1+2
083A 260A              ROL M1+1
083C 2609              ROL M1
083E A508       NORM   LDA X1
0840 D0EE              BNE NORM1
0842 60         RTS1   RTS
0843            ;
0843            ;
0843 207F08     FSUB   JSR FCOMPL
0846 205608     SWPALG JSR ALGNSW
0849 A504       FADD   LDA X2
084B C508              CMP X1
084D D0F7              BNE SWPALG
```

```
084F 200008          JSR ADD
0852 50EA     ADDEND BVC NORM
0854 7005            BVS RTLOG
0856 90C4     ALGNSW BCC SWAP
0858 A509     FTAP   LDA M1
085A 0A              ASL
085B E608     RTLOG  INC X1
085D F075            BEQ OVFL
085F A2FA     RTLOG1 LDX #$FA
0861 760F     ROR1   ROR E+3,X
0863 E8              INX
0864 D0FB            BNE ROR1
0866 60              RTS
0867          ;
0867          ;
0867 200D08   FMUL   JSR MD1
086A 6508            ADC X1
086C 20BD08          JSR MD2
086F 18              CLC
0870 205F08   MUL1   JSR RTLOG1
0873 9003            BCC MUL2
0875 200008          JSR ADD
0878 88       MUL2   DEY
0879 10F5            BPL MUL1
087B 4603     MDEND  LSR SIGN
087D 90BF     NORMX  BCC NORM
087F 38       FCOMPL SEC
0880 A203            LDX #$03
0882 A900     COMPL1 LDA #$00
0884 F508            SBC X1,X
0886 9508            STA X1,X
0888 CA              DEX
0889 D0F7            BNE COMPL1
088B F0C5            BEQ ADDEND
088D          ;
088D 200D08   FDIV   JSR MD1
0890 E508            SBC X1
0892 20BD08          JSR MD2
0895 38       DIV1   SEC
0896 A202            LDX #$02
0898 B505     DIV2   LDA M2,X
089A F50C            SBC E,X
089C 48              PHA
089D CA              DEX
089E 10F8            BPL DIV2
08A0 A2FD            LDX #$FD
08A2 68       DIV3   PLA
08A3 9002            BCC DIV4
08A5 9508            STA M2+3,X
08A7 E8       DIV4   INX
08A8 D0F8            BNE DIV3
08AA 260B            ROL M1+2
08AC 260A            ROL M1+1
08AE 2609            ROL M1
08B0 0607            ASL M2+2
08B2 2606            ROL M2+1
08B4 2605            ROL M2
08B6 B01C            BCS OVFL
08B8 88              DEY
08B9 D0DA            BNE DIV1
08BB F0BE            BEQ MDEND
08BD 860B     MD2    STX M1+2
08BF 860A            STX M1+1
08C1 8609            STX M1
08C3 B00D            BCS OVCHK
08C5 3004            BMI MD3
08C7 68              PLA
08C8 68              PLA
08C9 90B2            BCC NORMX
```

```
08CB 4980        MD3    EOR #$80
08CD 8508               STA X1
08CF A017               LDY #$17
08D1 60                 RTS
08D2 10F7        OVCHK  BPL MD3
08D4 00          OVFL   BRK
08D5                    ;
08D5                    ;
08D5 205808      FIX1   JSR FTAP
08D8 A508        FIX    LDA X1
08DA 1013               BPL UNDEFL
08DC C98E               CMP #$8E
08DE D0F5               BNE FIX1
08E0 2409               BIT M1
08E2 100A               BPL FIXRTS
08E4 A50B               LDA M1+2
08E6 F006               BEQ FIXRTS
08E8 E60A               INC M1+1
08EA D002               BNE FIXRTS
08EC E609               INC M1
08EE 60          FIXRTS RTS
08EF A900        UNDEFL LDA #$00
08F1 8509               STA M1
08F3 850A               STA M1+1
08F5 60                 RTS
08F6                    ;
08F6                    ;
08F6                    ;*************************
08F6                    ;*                       *
08F6                    ;* LOG,LOG10 UND EXP     *
08F6                    ;* UNTERPROGRAMME        *
08F6                    ;*                       *
08F6                    ;*************************
08F6                    ;
08F6                    ;
08F6 A509        LOG    LDA M1
08F8 F002               BEQ ERROR
08FA 1001               BPL CONT
08FC 00          ERROR  BRK
08FD                    ;
08FD 201C08      CONT   JSR SWAP
0900 A200               LDX #$00
0902 A504               LDA X2
0904 A080               LDY #$80
0906 8404               STY X2
0908 4980               EOR #$80
090A 850A               STA M1+1
090C 1001               BPL SEXP2
090E CA                 DEX
090F 8609        SEXP2  STX M1
0911 202C08             JSR FLOAT
0914 A203               LDX #3
0916 B504        SEXP1  LDA X2,X
0918 9510               STA Z,X
091A B508               LDA X1,X
091C 9518               STA SEXP,X
091E BDCA09             LDA R22,X
0921 9508               STA X1,X
0923 CA                 DEX
0924 10F0               BPL SEXP1
0926 204308             JSR FSUB
0929 A203               LDX #3
092B B508        SAVET  LDA X1,X
092D 9514               STA T,X
092F B510               LDA Z,X
0931 9508               STA X1,X
0933 BDCA09             LDA R22.X
0936 9504               STA X2,X
0938 CA                 DEX
```

```
0939 10F0                 BPL SAVET
093B 204908               JSR FADD
093E A203                 LDX #3
0940 B514         TM2     LDA T,X
0942 9504                 STA X2,X
0944 CA                   DEX
0945 10F9                 BPL TM2
0947 208D08               JSR FDIV
094A A203                 LDX #3
094C B508         M1T     LDA X1,X
094E 9514                 STA T,X
0950 9504                 STA X2,X
0952 CA                   DEX
0953 10F7                 BPL M1T
0955 206708               JSR FMUL
0958 201C08               JSR SWAP
095B A203                 LDX #3
095D BDDA09       M1C     LDA C,X
0960 9508                 STA X1,X
0962 CA                   DEX
0963 10F8                 BPL M1C
0965 204308               JSR FSUB
0968 A203                 LDX #3
096A BDD609       M2MB    LDA MB,X
096D 9504                 STA X2,X
096F CA                   DEX
0970 10F8                 BPL M2MB
0972 208D08               JSR FDIV
0975 A203                 LDX #3
0977 BDD209       M2A1    LDA A1,X
097A 9504                 STA X2,X
097C CA                   DEX
097D 10F8                 BPL M2A1
097F 204908               JSR FADD
0982 A203                 LDX #3
0984 B514         M2T     LDA T,X
0986 9504                 STA X2,X
0988 CA                   DEX
0989 10F9                 BPL M2T
098B 206708               JSR FMUL
098E A203                 LDX #3
0990 BDDE09       M2MHL   LDA MHLF,X
0993 9504                 STA X2,X
0995 CA                   DEX
0996 10F8                 BPL M2MHL
0998 204908               JSR FADD
099B A203                 LDX #3
099D B518         LDEXP   LDA SEXP,X
099F 9504                 STA X2,X
09A1 CA                   DEX
09A2 10F9                 BPL LDEXP
09A4 204908               JSR FADD
09A7 A203                 LDX #3
09A9 BDCE09       MLE2    LDA LE2,X
09AC 9504                 STA X2,X
09AE CA                   DEX
09AF 10F8                 BPL MLE2
09B1 206708               JSR FMUL
09B4 60                   RTS
09B5              ;
09B5              ;
09B5              ;
09B5 20F608       LOG10   JSR LOG
09B8 A203                 LDX #3
09BA BDC609       L10     LDA LN10,X
09BD 9504                 STA X2,X
09BF CA                   DEX
09C0 10F8                 BPL L10
09C2 206708               JSR FMUL
```

```
09C5 60       273          RTS
09C6          274   ;
09C6 7E6F2D   275   LN10   HEX 7E6F2DED        ; 0.4342945
09C9 ED
09CA 805A82   276   R22    HEX 805A827A        ; 1.4142136
09CD 7A
09CE 7F58B9   277   LE2    HEX 7F58B90C        ; 0.69314718
09D1 0C
09D2 8052B0   278   A1     HEX 8052B040        ; 1.2920074
09D5 40
09D6 81AB86   279   MB     HEX 81AB8649        ; -2.6398577
09D9 49
09DA 806A08   280   C      HEX 806A0866        ; 1.6567626
09DD 66
09DE 7F4000   281   MHLF   HEX 7F400000        ; 0.5
09E1 00
09E2          282   ;
09E2          283   ;
09E2 A203     284   EXP    LDX #3
09E4 BDBA0A   285          LDA L2E,X
09E7 9504     286          STA X2,X
09E9 CA       287          DEX
09EA 10F8     288          BPL EXP+2
09EC 206708   289          JSR FMUL
09EF A203     290          LDX #3
09F1 B508     291   FSA    LDA X1,X
09F3 9510     292          STA Z,X
09F5 CA       293          DEX
09F6 10F9     294          BPL FSA
09F8 20D808   295          JSR FIX
09FB A50A     296          LDA M1+1
09FD 851C     297          STA INT
09FF 38       298          SEC
0A00 E97C     299          SBC #124
0A02 A509     300          LDA M1
0A04 E900     301          SBC #0
0A06 1015     302          BPL OVFLW
0A08 18       303          CLC
0A09 A50A     304          LDA M1+1
0A0B 6978     305          ADC #120
0A0D A509     306          LDA M1
0A0F 6900     307          ADC #0
0A11 100B     308          BPL CONTIN
0A13 A900     309          LDA #0
0A15 A203     310          LDX #3
0A17 9508     311   ZERO   STA X1,X
0A19 CA       312          DEX
0A1A 10FB     313          BPL ZERO
0A1C 60       314          RTS
0A1D          315   ;
0A1D 00       316   OVFLW  BRK
0A1E          317   ;
0A1E 202C08   318   CONTIN JSR FLOAT
0A21 A203     319          LDX #3
0A23 B510     320   ENTD   LDA Z,X
0A25 9504     321          STA X2,X
0A27 CA       322          DEX
0A28 10F9     323          BPL ENTD
0A2A 204308   324          JSR FSUB
0A2D A203     325          LDX #3
0A2F B508     326   ZSAV   LDA X1,X
0A31 9510     327          STA Z,X
0A33 9504     328          STA X2,X
0A35 CA       329          DEX
0A36 10F7     330          BPL ZSAV
0A38 206708   331          JSR FMUL
0A3B A203     332          LDX #3
0A3D BDBE0A   333   LA2    LDA A2,X
0A40 9504     334          STA X2,X
```

```
0A42 B508                LDA X1,X
0A44 9518                STA SEXP,X
0A46 CA                  DEX
0A47 10F4                BPL LA2
0A49 204908              JSR FADD
0A4C A203                LDX #3
0A4E BDC20A      LB2     LDA B2,X
0A51 9504                STA X2,X
0A53 CA                  DEX
0A54 10F8                BPL LB2
0A56 208D08              JSR FDIV
0A59 A203                LDX #3
0A5B B508        DLOAD   LDA X1,X
0A5D 9514                STA T,X
0A5F BDC60A              LDA C2,X
0A62 9508                STA X1,X
0A64 B518                LDA SEXP,X
0A66 9504                STA X2,X
0A68 CA                  DEX
0A69 10F0                BPL DLOAD
0A6B 206708              JSR FMUL
0A6E 201C08              JSR SWAP
0A71 A203                LDX #3
0A73 B514        LTMP    LDA T,X
0A75 9508                STA X1,X
0A77 CA                  DEX
0A78 10F9                BPL LTMP
0A7A 204308              JSR FSUB
0A7D A203                LDX #3
0A7F BDCA0A      LDD     LDA D,X
0A82 9504                STA X2,X
0A84 CA                  DEX
0A85 10F8                BPL LDD
0A87 204908              JSR FADD
0A8A 201C08              JSR SWAP
0A8D A203                LDX #3
0A8F B510        LFA     LDA Z,X
0A91 9508                STA X1,X
0A93 CA                  DEX
0A94 10F9                BPL LFA
0A96 204308              JSR FSUB
0A99 A203                LDX #3
0A9B B510        LF3     LDA Z,X
0A9D 9504                STA X2,X
0A9F CA                  DEX
0AA0 10F9                BPL LF3
0AA2 208D08              JSR FDIV
0AA5 A203                LDX #3
0AA7 BDDE09      LD12    LDA MHLF,X
0AAA 9504                STA X2,X
0AAC CA                  DEX
0AAD 10F8                BPL LD12
0AAF 204908              JSR FADD
0AB2 38                  SEC
0AB3 A51C                LDA INT
0AB5 6508                ADC X1
0AB7 8508                STA X1
0AB9 60                  RTS
0ABA 805C55      L2E     HEX 805C551E          ; 1.4426950509
0ABD 1E
0ABE 86576A      A2      HEX 86576AE1          ; 87.417497202
0AC1 E1
0AC2 894D3F      B2      HEX 894D3F1D          ; 617.9722695
0AC5 1D
0AC6 7B46FA      C2      HEX 7B46FA70          ; .03465735903
0AC9 70
0ACA 834FA3      D       HEX 834FA303          ; 9.9545957821
0ACD 03
0ACE                ;
                 FINI    END
```

```
0800- 18 A2 02 B5 09 75 05 95
0808- 09 CA 10 F7 60 06 03 20
0810- 12 08 24 09 10 05 20 7F
0818- 08 E6 03 38 A2 04 94 0B
0820- B5 07 B4 03 94 07 95 03
0828- CA D0 F3 60 A9 8E 85 08
0830- A5 09 C9 C0 30 0C C6 08
0838- 06 0B 26 0A 26 09 A5 08
0840- D0 EE 60 20 7F 08 20 56
0848- 08 A5 04 C5 08 D0 F7 20
0850- 00 08 50 EA 70 05 90 C4
0858- A5 09 0A E6 08 F0 75 A2
0860- FA 76 0F E8 D0 FB 60 20
0868- 0D 08 65 08 20 BD 08 18
0870- 20 5F 08 90 03 20 00 08
0878- 88 10 F5 46 03 90 BF 38
0880- A2 03 A9 00 F5 08 95 08
0888- CA D0 F7 F0 C5 20 0D 08
0890- E5 08 20 BD 08 38 A2 02
0898- B5 05 F5 0C 48 CA 10 F8
08A0- A2 FD 68 90 02 95 08 E8
08A8- D0 F8 26 0B 26 0A 26 09
08B0- 06 07 26 06 26 05 B0 1C
08B8- 88 D0 DA F0 BE 86 0B 86
08C0- 0A 86 09 B0 0D 30 04 68
08C8- 68 90 B2 49 80 85 08 A0
08D0- 17 60 10 F7 00 20 58 08
08D8- A5 08 10 13 C9 8E D0 F5
08E0- 24 09 10 0A A5 0B F0 06
08E8- E6 0A D0 02 E6 09 60 A9
08F0- 00 85 09 85 0A 60
*
```

7.53 Hexdump FADD, FSUB, FMUL, FDIV

```
08F6- A5 09
08F8- F0 02 10 01 00 20 1C 08
0900- A2 00 A5 04 A0 80 84 04
0908- 49 80 85 0A 10 01 CA 86
0910- 09 20 2C 08 A2 03 B5 04
0918- 95 10 B5 08 95 18 BD CA
0920- 09 95 08 CA 10 F0 20 43
0928- 08 A2 03 B5 08 95 14 B5
0930- 10 95 08 BD CA 09 95 04
0938- CA 10 F0 20 49 08 A2 03
0940- B5 14 95 04 CA 10 F9 20
0948- 8D 08 A2 03 B5 08 95 14
0950- 95 04 CA 10 F7 20 67 08
0958- 20 1C 08 A2 03 BD DA 09
0960- 95 08 CA 10 F8 20 43 08
0968- A2 03 BD D6 09 95 04 CA
0970- 10 F8 20 8D 08 A2 03 BD
0978- D2 09 95 04 CA 10 F8 20
0980- 49 08 A2 03 B5 14 95 04
0988- CA 10 F9 20 67 08 A2 03
0990- BD DE 09 95 04 CA 10 F8
0998- 20 49 08 A2 03 B5 18 95
09A0- 04 CA 10 F9 20 49 08 A2
09A8- 03 BD CE 09 95 04 CA 10
09B0- F8 20 67 08 60 20 F6 08
09B8- A2 03 BD C6 09 95 04 CA
09C0- 10 F8 20 67 08 60 7E 6F
09C8- 2D ED 80 5A 82 7A 7F 58
09D0- B9 0C 80 52 B0 40 81 AB
09D8- 86 49 80 6A 08 66 7F 40
09E0- 00 00
*
```

7.54 Hexdump LOG, LOG10

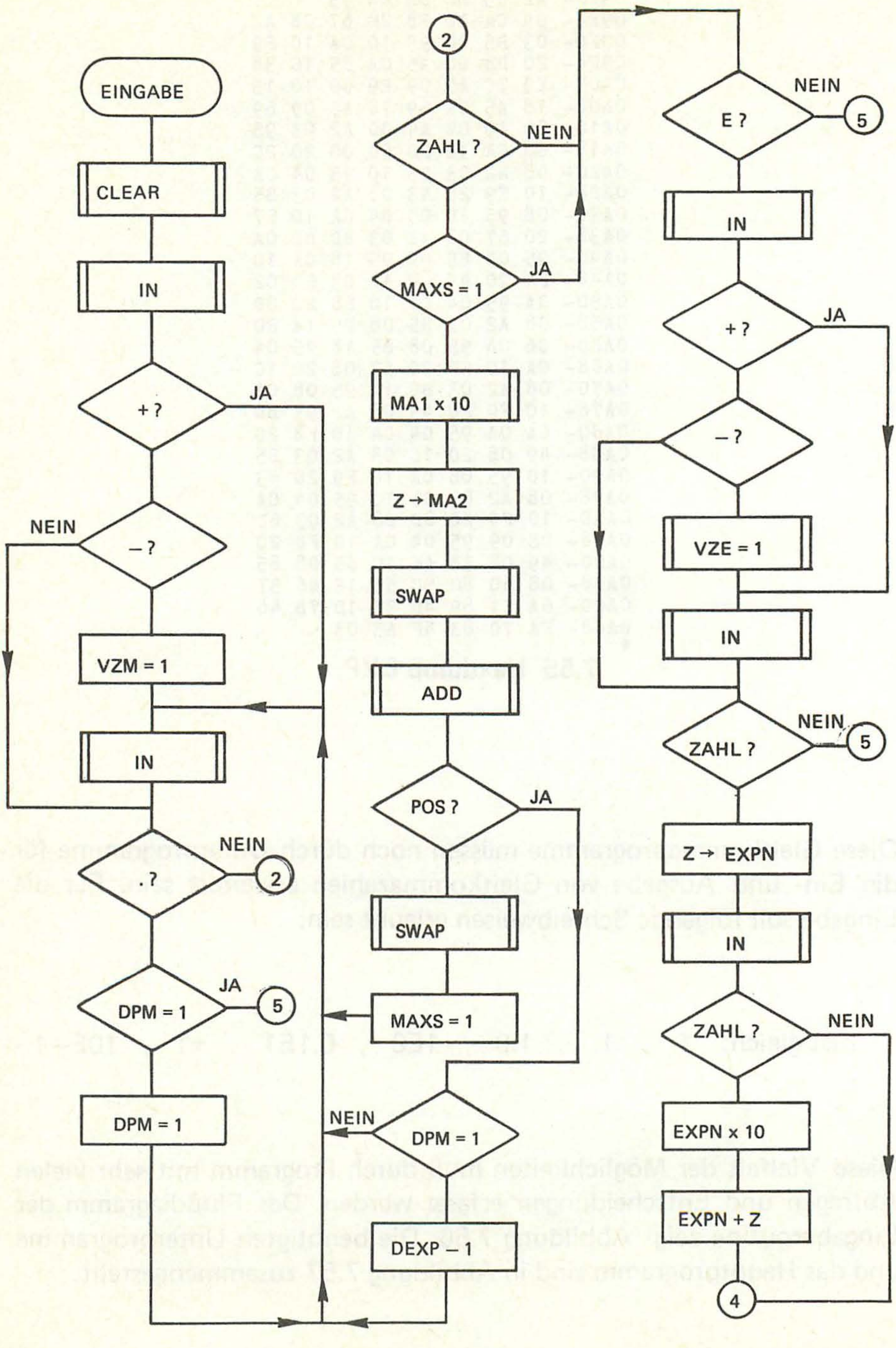
```
09E2- A2 03 BD BA 0A 95
09E8- 04 CA 10 F8 20 67 08 A2
09F0- 03 B5 08 95 10 CA 10 F9
09F8- 20 D8 08 A5 0A 85 1C 38
0A00- E9 7C A5 09 E9 00 10 15
0A08- 18 A5 0A 69 78 A5 09 69
0A10- 00 10 0B A9 00 A2 03 95
0A18- 08 CA 10 FB 60 00 20 2C
0A20- 08 A2 03 B5 10 95 04 CA
0A28- 10 F9 20 43 08 A2 03 B5
0A30- 08 95 10 95 04 CA 10 F7
0A38- 20 67 08 A2 03 BD BE 0A
0A40- 95 04 B5 08 95 18 CA 10
0A48- F4 20 49 08 A2 03 BD C2
0A50- 0A 95 04 CA 10 F8 20 8D
0A58- 08 A2 03 B5 08 95 14 BD
0A60- C6 0A 95 08 B5 18 95 04
0A68- CA 10 F0 20 67 08 20 1C
0A70- 08 A2 03 B5 14 95 08 CA
0A78- 10 F9 20 43 08 A2 03 BD
0A80- CA 0A 95 04 CA 10 F8 20
0A88- 49 08 20 1C 08 A2 03 B5
0A90- 10 95 08 CA 10 F9 20 43
0A98- 08 A2 03 B5 10 95 04 CA
0AA0- 10 F9 20 8D 08 A2 03 BD
0AA8- DE 09 95 04 CA 10 F8 20
0AB0- 49 08 38 A5 1C 65 08 85
0AB8- 08 60 80 5C 55 1E 86 57
0AC0- 6A E1 89 4D 3F 1D 7B 46
0AC8- FA 70 83 4F A3 03
*
```

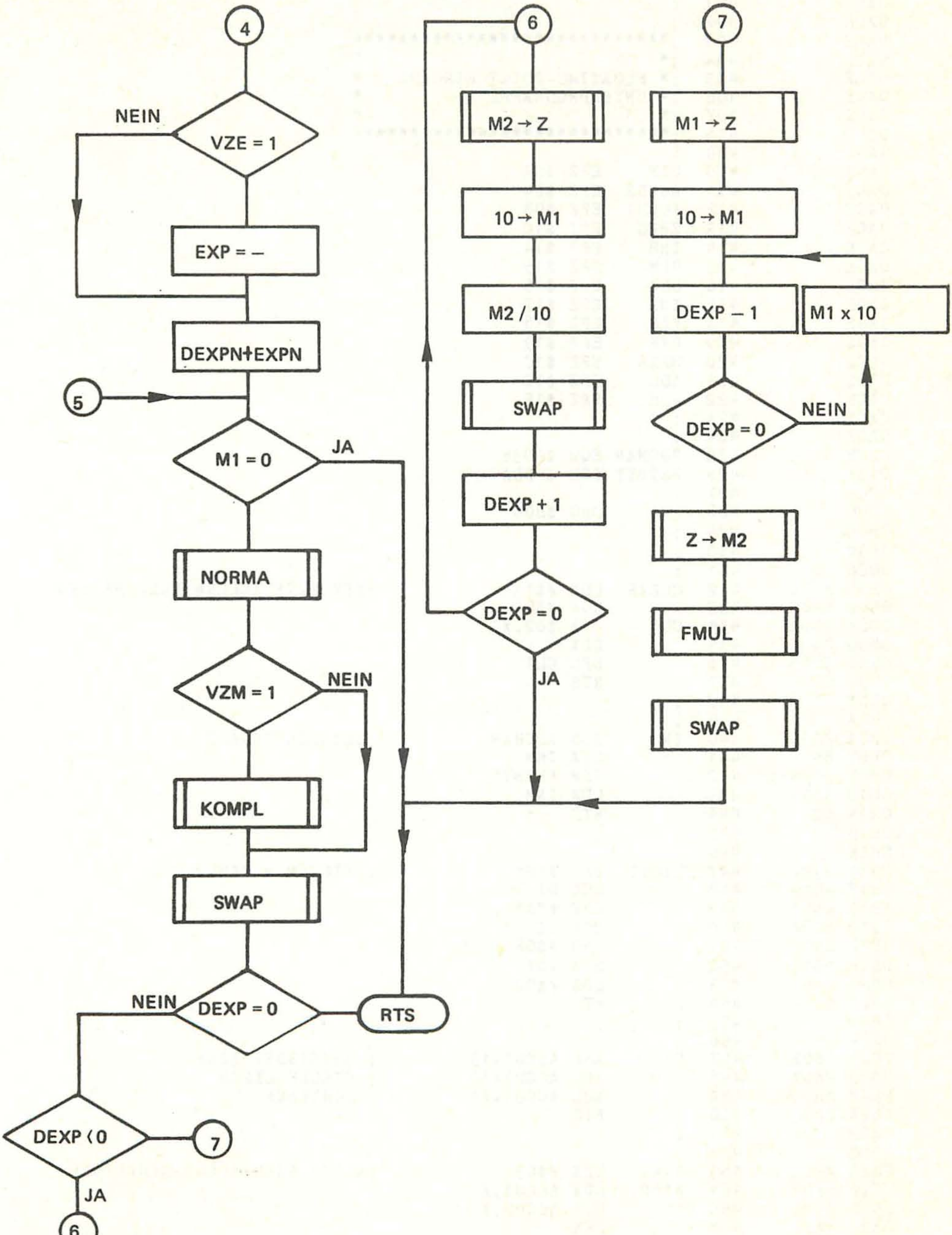
7.55 Hexdump EXP.

Diese Gleitkommaprogramme müssen noch durch Unterprogramme für die Ein- und Ausgabe von Gleitkommazahlen erweitert sein. Für die Eingabe soll folgende Schreibweisen erlaubt sein:

1 ist gleich: 1 , 1. , 1.0 , 1E0 , 0.1E1 , +1 , 10E–1

Diese Vielfalt der Möglichkeiten muß durch Programm mit sehr vielen Abfragen und Entscheidungen erfasst werden. Das Flußdiagramm der Eingaberoutine zeigt Abbildung 7.56. Die benötigten Unterprogramme und das Hauptprogramm sind in Abbildung 7.57 zusammengestellt.


EINGABE
CLEAR
IN
+ ?
JA
NEIN
– ?
VZM = 1
IN
NEIN
. ?
2
JA
DPM = 1
5
DPM = 1
2
ZAHL ?
NEIN
JA
MAXS = 1
MA1 x 10
Z → MA2
SWAP
ADD
POS ?
JA
SWAP
MAXS = 1
NEIN
DPM = 1
DEXP – 1
E ?
NEIN
5
IN
+ ?
JA
– ?
VZE = 1
IN
ZAHL ?
NEIN
5
Z → EXPN
IN
ZAHL ?
NEIN
EXPN x 10
EXPN + Z
4



7.56 Flußdiagramm Eingabe

```
0ACE            ;
0ACE            ;
0ACE            ;*****************************
0ACE            ;*                           *
0ACE            ;* FLOATING-POINT EINGABE    *
0ACE            ;* UNTERPROGRAMME            *
0ACE            ;*                           *
0ACE            ;*****************************
0ACE            ;
0ACE            VZM    EPZ $03
0ACE            ACCU2  EPZ $04
0ACE            ACCU1  EPZ $08
0ACE            ZREG   EPZ $10
0ACE            INM    EP7 $14
0ACE            DPM    EPZ $15
0ACE            DEX    EPZ $16
0ACE            INZ    EPZ $17
0ACE            VZE    EPZ $18
0ACE            EXP    EPZ $19
0ACE            MAXS   EPZ $1C
0ACE            ADL    EPZ $1E
0ACE            ADH    EPZ $1F
0ACE            ;
0ACE            ;
0ACE            RDCHAR EQU $FD35
0ACE            PRTBYT EQU $FDDA
0ACE            ;
0B00                   ORG $B00
0B00            ;
0B00            ;
0B00            ;
0B00 A21A       CLEAR  LDX #$1A          ;BENUTZTE ZELLEN NULL SETZEN
0B02 A900              LDA #$00
0B04 9502       CL1    STA $02,X
0B06 CA                DEX
0B07 10FB              BPL CL1
0B09 60                RTS
0B0A            ;
0B0A            ;
0B0A 2035FD     IN     JSR RDCHAR        ;ZEICHENEINGABE
0B0D 8514              STA INM
0B0F 20DAFD            JSR PRTBYT
0B12 A514              LDA INM
0B14 60                RTS
0B15            ;
0B15            ;
0B15 C9B0       DIGIT  CMP #"0"          ;ZEICHEN = ZAHL?
0B17 900A              BCC DI
0B19 C9C1              CMP #"A"
0B1B B006              BCS DI
0B1D 290F              AND #$0F
0B1F 8517              STA INZ
0B21 A900              LDA #$00
0B23 60         DI     RTS
0B24            ;
0B24            ;
0B24 060B       M1SL   ASL ACCU1+!3      ; MANTISSE1 EINE
0B26 260A              ROL ACCU1+!2      ; STELLE LINKS
0B28 2609              ROL ACCU1+!1      ; SCHIEBEN
0B2A 60                RTS
0B2B            ;
0B2B            ;
0B2B A203       A1A2   LDX #$03          ;ACCU1 NACH ACCU2 SCHREIBEN
0B2D B508       A100   LDA ACCU1,X
0B2F 9504              STA ACCU2,X
0B31 CA                DEX
0B32 10F9              BPL A100
0B34 60                RTS
0B35            ;
```

```
0B35 202B0B   471  MULTEN JSR A1A2             ;MANTISSE1 MIT 10 MULTIPLIZIEREN
0B38 20240B   472         JSR M1SL
0B3B 20240B   473         JSR M1SL
0B3E 200008   474         JSR ADD
0B41 4C240B   475         JMP M1SL
0B44          476  ;
0B44          477  ;
0B44 A996     478  NORMA  LDA #$96
0B46 8508     479         STA ACCU1
0B48 4C3E08   480         JMP NORM
0B4B          481  ;
0B4B          482  ;
0B4B A509     483  NULL   LDA M1
0B4D D006     484         BNE NU1
0B4F A50A     485         LDA M1+!1
0B51 D002     486         BNE NU1
0B53 A50B     487         LDA M1+!2
0B55 60       488  NU1    RTS
0B56          489  ;
0B56          490  ;
0B56 A203     491  A2Z    LDX #$03            ;ACCU2 NACH ZREG
0B58 B504     492  A21    LDA ACCU2,X
0B5A 9510     493         STA ZREG,X
0B5C CA       494         DEX
0B5D 10F9     495         BPL A21
0B5F 60       496         RTS
0B60          497  ;
0B60 A203     498  ZA2    LDX #$03            ;ZREG NACH ACCU2
0B62 B510     499  ZA21   LDA ZREG,X
0B64 9504     500         STA ACCU2,X
0B66 CA       501         DEX
0B67 10F9     502         BPL ZA21
0B69 60       503         RTS
0B6A          504  ;
0B6A A900     505  TEN    LDA #$00
0B6C 8509     506         STA M1
0B6E A90A     507         LDA #$0A
0B70 850A     508         STA M1+1
0B72 4C2C08   509         JMP FLOAT
0B75          510  ;
0B75 A000     511  STORE  LDY #$00
0B77 B90800   512  ST1    LDA ACCU1,Y
0B7A 911E     513         STA (ADL),Y
0B7C C8       514         INY
0B7D C004     515         CPY #$04
0B7F D0F6     516         BNE ST1
0B81 60       517         RTS
0B82          518  ;
0B82 A000     519  LOAD   LDY #$00
0B84 B11E     520  LO1    LDA (ADL),Y
0B86 990800   521         STA ACCU1,Y
0B89 C8       522         INY
0B8A C004     523         CPY #$04
0B8C D0F6     524         BNE LO1
0B8E 60       525         RTS
0B8F 60       528         RTS
0B90          529  ;
0B90          530  ;
0B90          531  ;
0B90          532  ;
0B90          533  ;******************************
0B90          534  ;*                            *
0B90          535  ;* TASTENFELD-EINGABE NACH    *
0B90          536  ;* ACCU2.                     *
0B90          537  ;*                            *
0B90          538  ;******************************
0B90          539  ;
0B90          540  ;
0B90 20000B   541  EIN    JSR CLEAR
```

```
0B93 200A0B         JSR IN
0B96 C9AB           CMP #"+"
0B98 F006           BEQ EI1
0B9A C9AD           CMP #"-"
0B9C D005           BNE EI2
0B9E E603           INC VZM
0BA0 200A0B   EI1   JSR IN
0BA3 C9AE     EI2   CMP #"."
0BA5 D008           BNE EI3
0BA7 A515           LDA DPM
0BA9 D075           BNE EI10
0BAB E615           INC DPM
0BAD 10F1           BPL EI1
0BAF 20150B   EI3   JSR DIGIT
0BB2 D025           BNE EI5
0BB4 A51C           LDA MAXS
0BB6 D021           BNE EI5
0BB8 20350B         JSR MULTEN
0BBB 1007           BPL EI4
0BBD 201C08         JSR SWAP
0BC0 E61C           INC MAXS
0BC2 10DC           BPL EI1
0BC4 A517     EI4   LDA INZ
0BC6 8507           STA M2+2
0BC8 A900           LDA #$00
0BCA 8506           STA M2+1
0BCC 8505           STA M2
0BCE 200008         JSR ADD
0BD1 A515           LDA DPM
0BD3 F0CB           BEQ EI1
0BD5 C616           DEC DEX
0BD7 30C7           BMI EI1
0BD9 EA       EI5   NOP
0BDA EA             NOP
0BDB C9C5           CMP #"E"
0BDD D041           BNE EI10
0BDF 200A0B         JSR IN
0BE2 C9AB           CMP #"+"
0BE4 F006           BEQ EI6
0BE6 C9AD           CMP #"-"
0BE8 D005           BNE EI7
0BEA E618           INC VZE
0BEC 200A0B   EI6   JSR IN
0BEF 20150B   EI7   JSR DIGIT
0BF2 D02C           BNE EI10
0BF4 A517           LDA INZ
0BF6 8519           STA EXP
0BF8 200A0B         JSR IN
0BFB 20150B         JSR DIGIT
0BFE D00D           BNE EI8
0C00 A519           LDA EXP
0C02 0A             ASL
0C03 0A             ASL
0C04 18             CLC
0C05 6519           ADC EXP
0C07 0A             ASL
0C08 18             CLC
0C09 6517           ADC INZ
0C0B 8519           STA EXP
0C0D A618     EI8   LDX VZE
0C0F F008           BEQ EI9
0C11 A519           LDA EXP
0C13 49FF           EOR #$FF
0C15 8519           STA EXP
0C17 E619           INC EXP
0C19 A516     EI9   LDA DEX
0C1B 18             CLC
0C1C 6519           ADC EXP
0C1E 8516           STA DEX
```

```
0C20 20440B   EI10   JSR NORMA
0C23 A503            LDA VZM
0C25 F003            BEQ EI11
0C27 207F08          JSR FCOMPL
0C2A EA       EI11   NOP
0C2B A900            LDA #$00
0C2D 8507            STA M2+2
0C2F 201C08          JSR SWAP
0C32 A516            LDA DEX
0C34 F034            BEQ EI15
0C36 301F            BMI EI14
0C38 20560B          JSR A2Z
0C3B 206A0B          JSR TEN
0C3E 202B0B          JSR A1A2
0C41 C616     EI12   DEC DEX
0C43 F006            BEQ EI13
0C45 206708          JSR FMUL
0C48 18              CLC
0C49 90F6            BCC EI12
0C4B 20600B   EI13   JSR ZA2
0C4E 206708          JSR FMUL
0C51 201C08          JSR SWAP
0C54 18              CLC
0C55 9013            BCC EI15
0C57 20560B   EI14   JSR A2Z
0C5A 206A0B          JSR TEN
0C5D 208D08          JSR FDIV
0C60 201C08          JSR SWAP
0C63 E616            INC DEX
0C65 F003            BEQ EI15
0C67 18              CLC
0C68 90ED            BCC EI14
0C6A A900     EI15   LDA #$00
0C6C 851C            STA MAXS
0C6E 60              RTS
0C6F                 ;
0C6F                 ;
              FINI   END

***** END OF ASSEMBLY
```

5.57 Gleitkommaeingabe

```
BR
*B00.C6F

0B00- A2 1A A9 00 95 02 CA 10
0B08- FB 60 20 35 FD 85 14 20
0B10- DA FD A5 14 60 C9 B0 90
0B18- 0A C9 C1 B0 06 29 0F 85
0B20- 17 A9 00 60 06 0B 26 0A
0B28- 26 09 60 A2 03 B5 08 95
0B30- 04 CA 10 F9 60 20 2B 0B
0B38- 20 24 0B 20 24 0B 20 00
0B40- 08 4C 24 0B A9 96 85 08
0B48- 4C 3E 08 A5 09 D0 06 A5
0B50- 0A D0 02 A5 0B 60 A2 03
0B58- B5 04 95 10 CA 10 F9 60
0B60- A2 03 B5 10 95 04 CA 10
0B68- F9 60 A9 00 85 09 A9 0A
0B70- 85 0A 4C 2C 08 A0 00 B9
0B78- 08 00 91 1E C8 C0 04 D0
0B80- F6 60 A0 00 B1 1E 99 08
0B88- 00 C8 C0 04 D0 F6 60 60
0B90- 20 00 0B 20 0A 0B C9 AB
0B98- F0 06 C9 AD D0 05 E6 03
```

```
0BA0- 20 0A 0B C9 AE D0 08 A5
0BA8- 15 D0 75 E6 15 10 F1 20
0BB0- 15 0B D0 25 A5 1C D0 21
0BB8- 20 35 0B 10 07 20 1C 08
0BC0- E6 1C 10 DC A5 17 85 07
0BC8- A9 00 85 06 85 05 20 00
0BD0- 08 A5 15 F0 CB C6 16 30
0BD8- C7 EA EA C9 C5 D0 41 20
0BE0- 0A 0B C9 AB F0 06 C9 AD
0BE8- D0 05 E6 18 20 0A 0B 20
0BF0- 15 0B D0 2C A5 17 85 19
0BF8- 20 0A 0B 20 15 0B D0 0D
0C00- A5 19 0A 0A 18 65 19 0A
0C08- 18 65 17 85 19 A6 18 F0
0C10- 08 A5 19 49 FF 85 19 E6
0C18- 19 A5 16 18 65 19 85 16
0C20- 20 44 0B A5 03 F0 03 20
0C28- 7F 08 EA A9 00 85 07 20
0C30- 1C 08 A5 16 F0 34 30 1F
0C38- 20 56 0B 20 6A 0B 20 2B
0C40- 0B C6 16 F0 06 20 67 08
0C48- 18 90 F6 20 60 0B 20 67
0C50- 08 20 1C 08 18 90 13 20
0C58- 56 0B 20 6A 0B 20 8D 08
0C60- 20 1C 08 E6 16 F0 03 18
0C68- 90 ED A9 00 85 1C 60 4A
*
```

7.58 Hexdump Gleitkommaeingabe

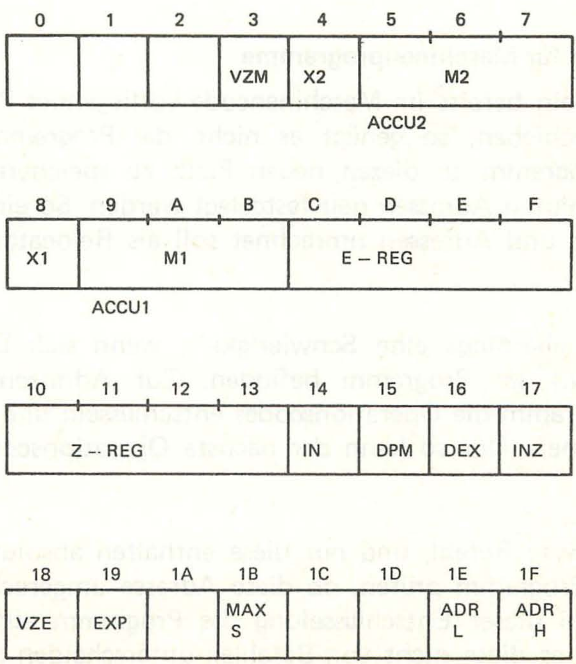
Die Eingabe beginnt mit dem Unterprogramm CLEAR, das alle benötigten Zellen in der Zero-Page löscht, und dem Unterprogramm IN, das ein Zeichen vom Tastenfeld liest und in einer Speicherzelle abgelegt. Ist dieses Zeichen ein Pluszeichen, dann kann sofort das nächste Zeichen gelesen werden. Ist es ein Minuszeichen, so wird der Vorzeichenmerker der Mantisse VZM = 1 gesetzt. Auch hier kann sofort das nächste Zeichen gelesen werden. Wenn dieses Zeichen ein Punkt ist dann wird der Dezimalpunktmerker DPM = 1 gesetzt. Trifft dies alles nicht zu, dann gibt es noch 3 weitere Möglichkeiten. Wenn es eine Zahl ist, dies wird im Unterprogramm DIGIT festgestellt, dann wird diese Zahl in die Mantisse übernommen. Der alte Inhalt der Mantisse wird mit 10 multipliziert (MULTEN) und die neu eingegebene Zahl hinzu addiert. Die alte Mantisse wird allerdings zwischengespeichert, denn wenn bei dieser Multiplikation mit anschließender Addition ein Übertrag in das Vorzeichenbit der Mantisse erfolgt (siehe Abbildung 7.51), so sind zu viele Stellen eingegeben worden. Dann wird MAXS = 1 gesetzt und die alte Mantisse wieder übernommen. Dadurch können noch Zahlen eingegeben werden, sie werden aber nicht mehr berücksichtigt.

Bleibt die Mantisse positiv, dann wird, wenn der Dezimalpunktmerker

DPM = 1 ist, von Dezimalexponenten 1 abgezogen, da es sich um eine Stelle hinter dem Komma handelt.

Jetzt verbleiben noch 2 Möglichkeiten. Ist das Zeichen ein E, so folgt die Eingabe des Exponenten. Sind alle Abfragen negativ verlaufen, so handelt es sich um ein beliebiges Zeichen, das die Eingabe beendet. Dann wird über die Schnittstelle 5 die Normalisierung der Zahl eingeleitet.

Dort wird zuerst gefragt, ob die Mantisse Null ist. Wenn ja, dann ist eine Null eingegeben worden und die Eingabe ist beendet. Wenn nicht, wird die Mantisse normalisiert und, wenn es sich um eine negative Zahl handelt, komplementiert. Dazu werden die Unterprogramme aus den Gleitkommaprogrammen benutzt. Nach der Eingabe soll die Zahl im Akkumulator 2 sein, deshalb wird mit SWAP die Zahl dort hingeschrieben. Eine Übersicht über die belegten Zellen und ihre Namen zeigt Abbildung 7.59.



7.59 Speicherbelegung für die FP Eingabeprogramme

Ist der Dezimalexponent Null, so ist auch die Eingabe beendet. Ist er kleiner als Null, so ist die Gleitkommazahl zu groß. Sie wird solange durch 10 dividiert, bis DEXP = 0 ist. Ist DEXP größer Null, so ist die Zahl zu klein, sie wird dann wiederum solange mit 10 multipliziert bis DEXP = 0 ist. Damit ist die Eingabe beendet.

Dieses Beispiel der Eingabe einer Gleitkommazahl zeigt, wie umfangreich Eingabeprogramme werden können, wenn alle Möglichkeiten berücksichtigt werden sollen.

Für den Leser sei es überlassen, die Ausgabe einer Gleitkommazahl als Programmieraufgabe zu lösen.

Bei den Unterprogrammen für die Eingabe befinden sich noch 2 Programme LOAD und STORE die eine Gleitkommazahl in ACCU2 über einen Vektor in den Zellen $1E, $1F holen oder wegspeichern.

7.6 Relocator für Maschinenprogramme

Möchte man ein bereits im Maschinencode vorliegendes Programm im Speicher verschieben, so genügt es nicht, das Programm durch ein Verschiebeprogramm an diesen neuen Platz zu speichern, es müssen auch alle absoluten Adressen neu festgelegt werden. So ein Programm, das verschiebt und Adressen umrechnet soll als Relocator bezeichnet werden.

Hier besteht allerdings eine Schwierigkeit, wenn sich Daten (Text, Tabellen usw.) im Programm befinden. Zur Adressenumrechnung muß das Programm die Operationscodes entschlüsseln und die Befehlslänge bestimmen. Nur so kann der nächste Operationscode gefunden werden.

Ist es ein 3-Byte Befehl, und nur diese enthalten absolute Adressen, so muß das Programm prüfen, ob diese Adresse umgerechnet werden soll. Trifft bei dieser Entschlüsselung das Programm auf ein Datenfeld, so kann es diese nicht von Befehlen unterscheiden und versucht sie ebenfalls zu entschlüsseln und umzurechnen. Dabei kann aber unter Umständen die ganze Programmstruktur verändert werden. Im Kapitel 9

"Disassemblierung" wird dies noch an Beispielen gezeigt.

Das Programm benutzt folgende Speicherzellen der Zero-Page:

RFLAG	0	=0	Verschieben und Umrechnen
		=1	Nur Verschieben
TEST1	1	LSB	Untere Grenze für Adressumrechnungen
	2	MSB	
TEST2	3	LSB	Obere Grenze für Adressumrechnungen
	4	MSB	
START	5	LSB	Anfang
	6	MSB	
STOP	7	LSB	Ende des zu verschiebenden
	8	MSB	Programms
OPTR	9	LSB	Zieladresse
	$A	MSB	
TEMP2	$D		
NPTR	$F		Hilfszellen
TEMP1	$11		

Um nun den vorher beschriebenen Konflikt zwischen echten Befehlen und Daten zu vermeiden wird vom Programm der Operationscode 00 (BRK) nicht übertragen. Damit kann man nun Programme mit Daten folgendermaßen umsetzen. Mit RFLAG = 1 wird das Programm an die neue Stelle geschrieben. Dabei findet keine Umrechnung statt. Im Ursprungsprogramm werden alle Daten durch 00 ersetzt und das Programm mit Adressumrechnung (RFLAG = 0) verschoben. Dabei werden die Nullen nicht übertragen und die schon vorher übertragenen Daten bleiben erhalten. Mit folgender Tabelle verschiebt sich das Programm selbst von $800 nach $900

0: 00 , 00 , 08 , F2 , 08 , 00 , 08 , F2 , 08 , 00 , 09

Das Programm mit Hexdump zeigt Abbildung 7.61.

Im Programm wird bei einigen Befehlen die indiziert indirekte Adressierung verwendet LDA (OPTR,X). Da das Indexregister Null ist, entspricht dies auch einer normalen indirekten Adressierung. Programm aus DDJ, April 1977.

```
0800           ;
0800           ;
0800           ;*****************************
0800           ;*                           *
0800           ;*   PROGRAMM RELOCATOR      *
0800           ;*   FUER 6502 CPU
0800           ;*                           *
0800           ;*****************************
0800           ;
0800           ;
0800           RFLAG  EPZ $0
0800           TEST1  EPZ $1
0800           TEST2  EPZ $3
0800           START  EPZ $5
0800           STOP   EPZ $7
0800           BEG    EPZ $9
0800           OPTR   EPZ $B
0800           TEMP2  EPZ $D
0800           NPTR   EPZ $F
0800           TEMP1  EPZ $11
0800           ;
0800           ;
0800 A205      BEGIN  LDX #$5
0802 B505      S10    LDA START,X
0804 950B             STA OPTR,X
0806 CA               DEX
0807 10F9             BPL S10
0809 E8               INX
080A A500      MOVE   LDA RFLAG              ;CHECK FOR DATA MOVE
080C F006             BEQ MO1
080E 204F08           JSR MOV1
0811 4C6008           JMP DONE?
0814 A10B      MO1    LDA (OPTR,X)
0816 A8               TAY
0817 D006             BNE MO2
0819 205308           JSR SKIP
081C 4C6008           JMP DONE?
081F 204F08    MO2    JSR MOV1
0822 98        TSTJSR TYA
0823 C920             CMP #$20
0825 D003             BNE BYTE1
0827 4C7A08           JMP BYTE3
082A           ;TEST FOR 1 BYTE INSTRUCTION
082A 98        BYTE1  TYA
082B 299F             AND #$9F
082D F031             BEQ DONE?
082F 98               TYA
0830 291D             AND #$1D
0832 C908             CMP #$8
0834 F02A             BEQ DONE?
0836 C918             CMP #$18
0838 F026             BEQ DONE?
083A           ;TEST FOR 3 BYTE INSTRUCTION
083A           ;
083A 98               TYA
083B 291C             AND #$1C
083D C91C             CMP #$1C
083F F039             BEQ BYTE3
0841 C918             CMP #$18
0843 F035             BEQ BYTE3
```

```
0845 C90C           CMP #$0C
0847 F031           BEQ BYTE3
0849           ;
0849           ;REMAINING 2 BYTE INSTRUCTIONS
0849           ;
0849 204F08         JSR MOV1
084C 4C6008         JMP DONE?
084F           ;MOVE 1 BYTE
084F           ;
084F A10B      MOV1 LDA (OPTR,X)
0851 810F           STA (NPTR,X)
0853 20DA08    SKIP JSR IOPTR
0856 20E108         JSR INPTR
0859 60             RTS
085A           ;
085A           ;MOVE 2 BYTES
085A           ;
085A 204F08    MOV2 JSR MOV1
085D 204F08         JSR MOV1
0860 A50B      DONE? LDA OPTR
0862 8511           STA TEMP1
0864 A50C           LDA OPTR+1
0866 8512           STA TEMP1+1
0868 A507           LDA STOP
086A 850D           STA TEMP2
086C A508           LDA STOP+1
086E 850E           STA TEMP2+1
0870 20CF08         JSR TEST
0873 9095           BCC MOVE
0875 F093           BEQ MOVE
0877 00             BRK
0878 EA             NOP
0879 EA             NOP
087A           ;
087A           ;3 BYTE INSTRUCTIONS
087A           ;
087A A10B      BYTE3 LDA (OPTR,X)
087C 8511           STA TEMP1
087E 20DA08         JSR IOPTR
0881 A10B           LDA (OPTR,X)
0883 8512           STA TEMP1+1
0885 20E808         JSR DOPTR
0888 A501           LDA TEST1
088A 850D           STA TEMP2
088C A502           LDA TEST1+1
088E 850E           STA TEMP2+1
0890 20CF08         JSR TEST
0893 F002           BEQ B10
0895 90C3           BCC MOV2
0897 A503      B10  LDA TEST2
0899 850D           STA TEMP2
089B A504           LDA TEST2+1
089D 850E           STA TEMP2+1
089F 20CF08         JSR TEST
08A2 F002           BEQ B20
08A4 B0B4           BCS MOV2
08A6           ;
08A6           ;ADDRESS RECOMPUTATION
08A6           ;
08A6 38        B20  SEC
08A7 A10B           LDA (OPTR,X)
08A9 E505           SBC START
08AB 850D           STA TEMP2
08AD 20DA08         JSR IOPTR
08B0 A10B           LDA (OPTR,X)
08B2 E506           SBC START+1
08B4 850E           STA TEMP2+1
08B6 20DA08         JSR IOPTR
08B9 18             CLC
```

```
08BA A50D            LDA TEMP2
08BC 6509            ADC BEG
08BE 810F            STA (NPTR,X)
08C0 20E108          JSR INPTR
08C3 A50E            LDA TEMP2+1
08C5 650A            ADC BEG+1
08C7 810F            STA (NPTR,X)
08C9 20E108          JSR INPTR
08CC 4C6008          JMP DONE?
08CF            ;
08CF            ;TEST COMPARES 2 ADDRESSES
08CF            ;
08CF A512       TEST    LDA TEMP1+1
08D1 C50E            CMP TEMP2+1
08D3 D004            BNE T10
08D5 A511            LDA TEMP1
08D7 C50D            CMP TEMP2
08D9 60         T10     RTS
08DA            ;
08DA            ;INCREMENT OLD POINTER
08DA            ;
08DA E60B       IOPTR   INC OPTR
08DC D002            BNE INC10
08DE E60C            INC OPTR+1
08E0 60         INC10   RTS
08E1            ;
08E1            ;INCREMENT NEW POINTER
08E1            ;
08E1 E60F       INPTR   INC NPTR
08E3 D002            BNE INC20
08E5 E610            INC NPTR+1
08E7 60         INC20   RTS
08E8            ;
08E8            ;DECREMENT OLD POINTER
08E8            ;
08E8 C60B       DOPTR   DEC OPTR
08EA A50B            LDA OPTR
08EC C9FF            CMP #$FF
08EE D002            BNE D10
08F0 C60C            DEC OPTR+1
08F2 60         D10     RTS
                        END
```

```
0800- A2 05 B5 05 95 0B CA 10
0808- F9 E8 A5 00 F0 06 20 4F
0810- 08 4C 60 08 A1 0B A8 D0
0818- 06 20 53 08 4C 60 08 20
0820- 4F 08 98 C9 20 D0 03 4C
0828- 7A 08 98 29 9F F0 31 98
0830- 29 1D C9 08 F0 2A C9 18
0838- F0 26 98 29 1C C9 1C F0
0840- 39 C9 18 F0 35 C9 0C F0
0848- 31 20 4F 08 4C 60 08 A1
0850- 0B 81 0F 20 DA 08 20 E1
0858- 08 60 20 4F 08 20 4F 08
0860- A5 0B 85 11 A5 0C 85 12
0868- A5 07 85 0D A5 08 85 0E
0870- 20 CF 08 90 95 F0 93 00
0878- EA EA A1 0B 85 11 20 DA
0880- 08 A1 0B 85 12 20 E8 08
0888- A5 01 85 0D A5 02 85 0E
0890- 20 CF 08 F0 02 90 C3 A5
0898- 03 85 0D A5 04 85 0E 20
08A0- CF 08 F0 02 B0 B4 38 A1
08A8- 0B E5 05 85 0D 20 DA 08
08B0- A1 0B E5 06 85 0E 20 DA
08B8- 08 18 A5 0D 65 09 81 0F
```

```
08C0- 20 E1 08 A5 0E 65 0A 81
08C8- 0F 20 E1 08 4C 60 08 A5
08D0- 12 C5 0E D0 04 A5 11 C5
08D8- 0D 60 E6 0B D0 02 E6 0C
08E0- 60 E6 0F D0 02 E6 10 60
08E8- C6 0B A5 0B C9 FF D0 02
08F0- C6 0C 60
*
```

7.61 Programm und Hexdump RELOCATOR

7.7 Nützliche Tips und Unterprogramme

In diesem Abschnitt werden in bunter Folge einige Programmiertips und -tricks und nützliche Unterprogramme beschrieben.

Soll der Inhalt zweier Zellen daraufhin untersucht werden, ob dieser in beiden Zellen Null ist, so kann dies auf einfache Weise mit der ODER-Funktion geschehen.

```
LDA  ZELLE1
ORA  ZELLE2
BEQ  WEITER
 .
 .
 .
```

Nur wenn in beiden Zellen kein Bit 1 war, ist die BEQ-Bedingung erfüllt.

Die 6502 CPU hat nur einen indirekten Befehl:
JMP (ADR), wobei ADR eine Adresse in der Zero-Page ist. Benötigt man aber einen indirekten Sprung in ein Unterprogramm, aus dem man mit RTS im Hauptprogramm weitermachen möchte, so kann dies mit folgender Befehlsfolge geschehen. Irgendwo im Programm steht der Befehl

```
JUMPI   JMP (ADR)
```

Soll nun ein indirekter Sprung in ein Unterprogramm durchgeführt werden, so wird dessen Adresse nach ADR, ADR + 1 geschrieben und

im Programm ein JSR JUMPI aufgerufen. Das RTS führt dann auf den Befehl nach dem JSR zurück.

Im ATARI-Monitor findet man folgende Programmfolge für die Ausführung eines indirekten Sprunges

```
LDA  #ADRL
PHA
LDA  #ADRH
PHA
RTS
```

Die Adresse ADRL, ADRH wird mit PHA auf den Stapelspeicher geschrieben. Der Befehl RTS holt von dort seine Rücksprungadresse, so daß damit ein indirekter Sprung nach ADRL + 1 und ADRH ausgeführt wird. Wie schon in Abbildung 5.13 gezeigt, wird bei einem JSR nicht die Adresse des nächsten Befehls sondern die vorangehende Adresse im Stapel abgelegt.

Für Testzwecke benötigt man oft die Ausgabe des Inhalts einer oder mehrerer Speicherzellen als 2 stellige Hexadezimalzahl. Da im Monitor oft nur ein Unterprogramm für die Ausgabe eines ASCII-Zeichens vorhanden ist, kann dies mit folgendem Unterprogramm geschehen (entnommen dem KIM-Monitor). Das auszugebende Byte wird im Akkumulator an das Unterprogramm übergeben und ist nach Verlassen der Routine wieder im Akkumulator.

```
0800          ;
0800          ;
0800          ;*******************************
0800          ;*                             *
0800          ;*  AUSGABE EINES BYTES ALS    *
0800          ;*  ALS 2 HEXZAHLEN            *
0800          ;*                             *
0800          ;*******************************
0800          ;
0800          OUTCH  EQU $FDED
0800          ;
0800          ;
0800 8D2208   PRTBYT STA SAVE
0803 4A              LSR
0804 4A              LSR
0805 4A              LSR
0806 4A              LSR
0807 201408          JSR HEXTA
```

```
080A AD2208          LDA SAVE
080D 201408          JSR HEXTA
0810 AD2208          LDA SAVE
0813 60              RTS
0814         ;
0814         ;
0814 290F    HEXTA   AND #$0F        ; OBEREN 4 BIT AUSBLENDEN
0816 C90A            CMP #$0A        ;GROESSER 9?
0818 18              CLC
0819 3002            BMI HEXTA1
081B 6907            ADC #$07
081D 69B0    HEXTA1  ADC #$B0        ;NUR APPLE, SONST $30
081F 4CEDFD          JMP OUTCH
0822         ;
0822 00      SAVE    HEX 00
                     END
```

7.71 Programm PRTBYT

Das folgende Programm ist ein Speichertest. Nacheinander werden die Bitmuster % 1 0 1 0 1 0 1 0 und % 0 1 0 1 0 1 0 1 in eine Speicherzelle geschrieben und wieder ausgelesen. Wird dabei ein Unterschied festgestellt, so weist diese Zelle einen Hardwarefehler auf. Das Programm wird dann mit einem BRK-Befehl abgebrochen. Die Adresse der schadhaften Zelle ist dann in den Adressen $00 und $01 zu finden. Der Speichertest beginnt erst oberhalb des Stapels bei Adresse $200. Befindet sich ein Programm im Speicher, so wird dieses nicht zerstört, da der Zelleninhalt vor einem Text in der Zelle $03 gerettet und anschließend zurückgeschrieben wird. Nur der Inhalt der Zero-Pagezellen $00 bis $36 geht verloren.

```
0800          ;
0800          ;
0800          ;******************************
0800          ;*                            *
0800          ;* SPEICHERTEST               *
0800          ;*                            *
0800          ;******************************
0800          ;
0005                  ORG $05
0005                  OBJ $0800
0005          ;
0005 A000     TEST    LDY #$00
0007 8400             STY $00
0009 A902             LDA #$02
000B 8501             STA $01
000D B100     TST1    LDA ($00),Y
000F 8503             STA $03
0011 A9AA             LDA #$AA
0013 8502             STA $02
0015 9100             STA ($00),Y
0017 B100             LDA ($00),Y
0019 C502             CMP $02
```

```
001B D019           BNE FIN
001D A955           LDA #$55
001F 8502           STA $02
0021 9100           STA ($00),Y
0023 B100           LDA ($00),Y
0025 C502           CMP $02
0027 D00D           BNE FIN
0029 A503           LDA $03
002B 9100           STA ($00),Y
002D E600           INC $00
002F D002           BNE TST2
0031 E601           INC $01
0033 18       TST2  CLC
0034 90D7           BCC TST1
0036 00       FIN   BRK
0037          ;
                    END
```

7.72 Speichertest

Das nächste Programm erzeugt Zufallszahlen und Zufallsbuchstaben. Der Programmteil Zufallszahlen ist dem "First Book of KIM" entnommen. Im Programmteil Zufallsbuchstaben wird durch ZUFA eine Zufallszahl erzeugt und verglichen ob diese kleiner A (beim APPLE: $C1) oder größer Z (beim APPLE: $DB) ist. Wenn dies nicht der Fall ist, so wird die Zahl als ASCII-Zeichen ausgegeben. Durch Verändern der Startwerte in den Zellen $10 bis $15 werden andere Zufallsbuchstaben erzeugt.

```
0800          ;
0800          ;
0800          ;******************************
0800          ;*                            *
0800          ;* ZUFALLSBUCHSTABEN          *
0800          ;*                            *
0800          ;******************************
0800          ;
0800          ;
0800          OUTCH    EQU $FDED
0800          ;
0800 205008   TEST     JSR ZUFA
0803 C9C1              CMP #$C1
0805 90F9              BCC TEST
0807 C9DB              CMP #$DB
0809 B0F5              BCS TEST
080B 20EDFD            JSR OUTCH
080E 18                CLC
080F 90EF              BCC TEST
0811          ;
0811          ;
0811          ;
0811          ;******************************
0811          ;*                            *
0811          ;*  ZUFALLSZAHLEN             *
0811          ;*                            *
0811          ;******************************
0811          ;
0850                   ORG $850
```

```
0850          ;
0850          ZAHL     EPZ $10
0850          ;
0850 38       ZUFA     SEC
0851 A511              LDA ZAHL+1
0853 6514              ADC ZAHL+4
0855 6515              ADC ZAHL+5
0857 8510              STA ZAHL
0859 A204              LDX #$04
085B B510     ZU1      LDA ZAHL,X
085D 9511              STA ZAHL+1,X
085F CA                DEX
0860 10F9              BPL ZU1
0862 60                RTS
0863          ;
                       END

***** END OF ASSEMBLY
```

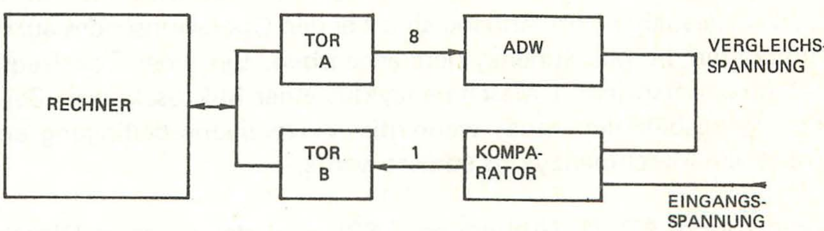
7.73 Zufallsbuchstaben

7.8 Zeitprobleme

In den meisten Fällen ist die Ausführungszeit von Programmen in Maschinencode sehr kurz, so daß keine besonderen Programmierkniffe angewendet werden müssen, um diese noch weiter zu verkürzen.

Dies soll jedoch auch an einem Beispiel gezeigt werden:

Mit einem 8-bit Digital-Analog Wandler soll durch ein Programm eine 8-bit Analog-Digital Wandlung ausgeführt werden.



7.81 Blockschaltbild Digital-Analog Wandlung

Ein Digital-Analog Wandler setzt eine Zahl, die an seinem Eingang anliegt, in einen Spannungswert um. Dieser Spannungswert wird durch einem Komparator mit der Eingangsspannung verglichen. Ist die Eingangsspannung größer als die Vergleichsspannung, so muß sie verkleinert werden. Vom Rechner wird die Zahl über ein Tor (siehe Kapitel

10) ausgegeben, über ein zweites Tor wird der Ausgang des Komparators von Rechner gelesen. Das Blockschaltbild einer solchen Wandelschaltung zeigt Abbildung 7.81.

Der Analog-Digital Wandler soll eine Ausgangsspannung von 0 bis 5 Volt besitzen. Wird an den Eingang $00 gelegt, so soll die Ausgangsspannung 0 Volt, mit $FF am Eingang soll die Ausgangsspannung 5 Volt sein.

Das Verfahren der Digital-Analog Wandlung läuft nun so ab, daß zuerst eine mittlere Spannung von 2,5 Volt, dies entspricht $80 am Eingang, mit der Eingangsspannung verglichen wird. Ist die Eingangsspannung größer, so wird dieser Spannungswert beibehalten und die Hälfte (1,25 Volt) hinzu addiert. $80 + $40 ergibt $C0 = 3.75 Volt. Ist die Eingangsspannung kleiner, so wird der Spannungswert halbiert und die Eingangsspannung damit verglichen. Auf diese Weise wird die Eingangsspannung in einen Zahlenwert umgewandelt. Bei den angegebenen Daten werden die 5 Volt mit einem Fehler von – 20 mV aufgelöst.

Für viele Fälle ist es nun vorteilhaft, wenn diese Analog-Digital Wandlung in einer möglichst kurzen Zeit durchgeführt wird.

An dieser Stelle ist es sinnvoll ein Programm auf seine Ausführungszeit zu untersuchen. Im Anhang sind bei den Operationscodes auch die Befehlszeiten in Maschinenzyklen angegeben. Bei einer Taktfrequenz von 1 MHz entspricht 1 Maschinenzyklus einer Mikrosekunde. Bei den Verzweigungsbefehlen muß, wenn die Verzweigungsbedingung erfüllt ist, noch ein Maschinenzyklus addiert werden.

Im Programm ADW1 (Abbildung 7.82) wird der normale Dienstweg beschritten. Als Zählregister wird das Indexregister X benutzt. $80 wird in einer Hilfszelle Z gespeichert und an das Tor A ausgegeben. Der Ausgang des Komparators wird über Bit 0 von Tor B eingelesen. Bit 0 ist 1, wenn die Eingangsspannung größer ist, als die Vergleichsspannung. Dann wird die Hälfte der verbleibenden Spannung dem augenblicklichen Wert hinzu addiert und ein neuer Vergleich durchgeführt. Ist dabei die Verglichsspannung zu groß, wird dieser Wert wieder abgezogen.

Programm ADW1

	LDX	#$08	2			
	LDA	#$80	2			
	STA	Z	3		7	7
M0	STA	TORA		4		
	LDA	TORB		4		
	AND	#$01		2		
	BEQ	M1		2 + 1	13	12
M2	LSR	Z		5		
	CLC			2		
	LDA	TORA		4		
	ADC	Z		3		
	DEX			2		
	BNE	M0		2 + 1	19	19
	RTS		6		6	6
M1	SEC			2		
	LDA	TORA		4		
	SBC	Z		3		
	CLC			2		
	BCC	M2		2 + 1	13	

Maximale Wandelzeit: 8 x 45 = 360
+ 13 = 373 μs

Minimale Wandelzeit: 8 x 31 = 248
+ 13 = 261 μs

7.82 Programm ADW1

Addiert man nun die Maschinenzyklen, so erhält man für die längste Schleife 45 Zyklen, für die kürzeste Schleife 31 Zyklen. Beide Schleifen werden 8 mal durchlaufen, sodaß dafür eine Maximalzeit von 360 μs und eine Minimalzeit von 248 μs erreicht wird. Dazu kommen noch 13 μs für das Setzen der Zellen am Anfang und für das RTS.

Dies ergibt eine mittlere Wandelzeit von 317 μs. Beim Programm ADW2 in Abbildung 7.83 wird nun durch Programmiertricks versucht, die Ausführungszeit zu verkürzen. Auf der Hardwareseite wird der Komparator nicht durch Bit 0 sondern durch Bit 7 von Tor B abgefragt. Dadurch kann das Ausblenden wegfallen und mit BPL oder BMI verzweigt werden. Das X-Register kann als Zählregister eben-

falls entfallen, denn wenn durch Linksschieben des Inhaltes von Z das Carry Bit gesetzt wird, dann ist die Umwandlung beendet. Der erste Vergleich wird mit $7F ausgeführt. Ist die Eingangsspannung größer dann wird mit ORA Z Bit 7 gesetzt. Mit EOR Z wird Bit 6 gelöscht, sodaß der nächste Vergleich mit $BF durchgeführt wird.

Die Maximalzeit ist hierfür 221 μs, die Minimalzeit 215 μs, und somit die mittlere Wandelzeit 218 μs.
Durch geschickte Programmierung kann **durchaus** Rechenzeit eingespart werden.

Programm ADW2

	LDA	#$80	2			
	STA	Z	3			
	LDA	#$7F	2		7	7
M0	STA	TORA		4		
	LDY	TORB		4		
	BPL	M1		2 + 1	11	10
	ORA	Z		3		
M1	LSR	Z		5		
	BCS	FIN		2 + 1		
	EOR	Z		3		
	BCC	M0		3	13	16
FIN	RTS			6		6

Maximale Wandelzeit: 8 x 26 = 208
+ 13 = 221 μs

Minimale Wandelzeit: 8 x 24 = 192
+ 13 = 215 μs

7.83 Programm ADW2

6502-Systeme

8.0 6502 Systeme

Die meisten der sogenannten Heim- oder Personalcomputer enthalten als CPU-Baustein den 6502. Der erste Einplatinencomputer, der Ahnvater dieser Familie, der auf den Markt kam, war der KIM 1. Durch die Verwendung eines normalen Kassettenrecorders mit handelsüblichen Tonkassetten zur Speicherung von Programmen stellte er den ersten kompletten Rechner zu einem sagenhaft niedrigen Preis dar. Bald darauf folgte als erster richtiger Computer mit Bildschirm und Tastatur bei den 6502 Systemen der PET und der APPLE I. Bei den Einplatinencomputern folgten der SYM, AIM 65, während PET sich zum CBM und APPLE I sich zum APPLE II erweiterte. Daneben entstanden die C1P bis C4P Systeme von Ohio Scientific. Nach dem APPLE III folgten nun als jüngste Kinder, die ATARI-Rechner 400 und 800 und der VC20 von Commodore. Auf einige dieser Systeme und ihre Eigenheiten bei der Programmierung in Maschinensprache soll nun eingegangen werden.

8.1 Der APPLE II-Computer

Bei diesem System werden zwei Assembler vorgestellt. Einmal der Miniassebler des Monitors und zweitens der Makro-Assembler des PASCAL-Systems.

8.11 Miniassembler des APPLE II-Computers

In Verbindung mit INTEGER BASIC und MONITOR verfügt der APPLE II-Computer über einen Miniassembler, der eine direkte Eingabe von Assemblerbefehlen ermöglicht. Die Befehle werden sofort entschlüsselt und gespeichert. Fehlermeldungen werden bei falscher Eingabe ausgegeben. Dieser Miniassembler kennt keine symbolischen

Namen. Die Adressenangabe bei Verzweigungsbefehlen erfolgt durch Angabe der absoluten Adresse. Diese ist nur bei Rückwärtsverzweigungen bekannt. Für Vorwärtsverzweigungen hilft man sich mit einem Trick. Man gibt eine fiktive Adresse an und ändert diese am Ende des Programms durch die richtigen Adressen ab.

Arbeitsablauf:

Aus Integer-BASIC mit CALL-151 in den Monitor, danach F666G. Als Kennzeichen erscheint am linken Rand des Bildschirms ein Ausrufezeichen (!). Nun wird die Startadresse, gefolgt von einem Doppelpunkt und der erste Assembler-Befehl eingegeben. Dieser wird bei Betätigen der RETURN-Taste übersetzt, die Adresse mit den Bitmustern ausgegeben und die Eingabe der nächsten Zeile durch ein Ausrufezeichen angezeigt.

Dabei braucht nun die Adresse nicht mehr angegeben werden, sie ist vom Rechner schon bestimmt worden. Die Eingabe einer Befehlszeile muß aber mit einem Leerzeichen beginnen. Abbildung 8.1 zeigt den Ausdruck des Bildschirms bei einer Programmierung mit dem Miniassembler.

```
*F666G

0900-   A9 00       LDA   #$00
0902-   85 10       STA   $10
0904-   A8          TAY
0905-   C8          INY
! STA (1000),Y
             ^
! STA (1000,Y)
            ^
0906-   8D 00 10    STA   $1000
0909-   99 00 10    STA   $1000,Y
090C-   B1 10       LDA   ($10),Y
090E-   C0 FF       CPY   #$FF
0910-   D0 F3       BNE   $0905
0912-   EA          NOP
0913-   EA          NOP
0914-   EA          NOP
0915-   A2 10       LDX   #$10
0917-   81 10       STA   ($10,X)
!
```

8.1 Miniassembler APPLE II

Besonderheiten:

Die Adressangabe wird immer als Hex-Zahl interpretiert, deshalb

braucht kein $—Zeichen vorangestellt werden.

Bei fehlerhafter Eingabe piepst er und der Cursor zeigt auf die Stelle, bei der ein Fehler entdeckt wurde. Der Ausstieg aus dem Miniassembler erfolgt entweder durch Reset oder durch ! $FF69G.

8.12 Der Makro-Assembler des PASCAL-Systems

Der Makro-Assembler des PASCAL-Systems dient hauptsächlich dazu, Maschinenprogramme in PASCAL einzubinden. Er verlangt deshalb vor dem Assemblerprogramm die Angabe, ob es such um eine Prozedur oder eine Funktion handelt, den Namen dieser Prozedur oder dieser Funktion und die Zahl der zu übergebenden Parameter. Wird der Assembler nur allein benötigt, wird vor dem Programm die Angabe PROC ⟨ NAME ⟩ verlangt.

Der Assembler kennt folgende Pseudocodes. Für die Schreibweise wollen wir folgende Vereinbarung treffen. Eine Angabe in spitzen Klammern ⟨ ⟩ muß erfolgen, eine Angabe in eckigen Klammern [ ] braucht nicht unbedingt gemacht werden.

[MARKE] . ASCII "⟨ ZEICHEN ⟩"
Beispiel: TEXT .ASCII "NEUE EINGABE"
Diese Anweisung speichert die angegebenen Zeichen ab.

[MARKE] .BYTE [WERTELISTE]
Beispiel: .BYTE 3
Damit wird für die Hexadezimalzahl 03 Speicherplatz reserviert. Wird kein Wert angegeben, so wird 00 abgespeichert.
Sollen mehrere verschiedene Byte reserviert werden, so sind diese durch Komma zu trennen. Beispiel: .BYTE 1,2,3,4
Bei der Ausgabe von Zahlen weicht dieser Assembler von der normalen Konvention ab, Hexadezimalzahlen durch ein $-Zeichen zu kennzeichnen. Eine Zahl beginnt immer mit einer Ziffer. Beginnt eine Hexadezimalzahl mit einem Buchstaben, so wird eine Null vorangestellt.
0FFFF entspricht $FFFF oder
0A9 entspricht $A9.
Soll eine Dezimalzahl angegeben werden so wird die Zahl, gefolgt von

einem Punkt geschrieben:

25. entspricht 25 Dezimal.

[MARKE] .BLOCK ⟨ LÄNGE ⟩ [, WERT]
Beispiel: PLATZ .BLOCK 5,2
Damit werden ab der Adresse PLATZ 5 Byte mit dem Inhalt 02 reserviert.

[MARKE] .WORD ⟨ WERTELISTE ⟩
Beispiel: .WORD 4,3,2,1
Ein Wort ist in PASCAL immer eine 16-bit Binärzahl. Deshalb wird die angegebene Werteliste als

0004
0003
0002
0001 gespeichert.

⟨ MARKE ⟩ .EQU ⟨ ADRESSE ⟩
Beispiel: TORA .EQU 0C0C0
Die EQUATE-Anweisung weist, wie schon besprochen Adressen symbolische Namen zu. Diese Adresse kann auch ein schon vorher vereinbarter symbolischer Name sein.

.ORG ⟨ WERT ⟩
Beispiel: .0RG 0A00
Die 0RG-Anweisung hat hier eine andere Bedeutung. Sie gibt nicht die absolute Adresse für den Maschinencode an, sondern den Abstand zum Beginn des assemblierten Programms. Dies kann durch die Pseudoanweisung

.ABSOLUTE

geändert werden. Dann werden alle .0RG-Anweisungen als absolute Adressen interpretiert. Die .ABSOLUTE Anweisung muß vor dem ersten .PROC erfolgen und gilt für die gesamte Assemblierung.

.END
Diese Pseudoanweisung beendet den Assemblertext.

Makrodefinition:

```
        .MACRO ⟨ NAME ⟩
       ⟨ TEXT ⟩
        .ENDM
```

Mit der Anweisung .MACRO ⟨ NAME ⟩ beginnt die Definition eines Makros. Dies ist Assemblertext der unter seinem Namen in anderen Assemblertexten aufgerufen werden kann. Die Definition eines Makros kann vor der Vereinbarung .PROC erfolgen. Ein Makro wird immer dann verwendet, wenn eine Befehlsfolge in Assemblerschreibweise sich an vielen Stellen des Programms wiederholt.

Dazu ein Beispiel:
Bei der Verarbeitung von Tabellen und Listen muß häufig der augenblickliche Wert einer Adresse mit einer Endadresse verglichen werden. Dazu benutzt man folgende Befehlsfolge:

```
        LDA  WERTH
        CMP  ENDH
        BCC  FIN
        BEQ  F
        BCS  FIN
F       LDA  WERTL
        CMP  ENDL
FIN     NOP
```

Tritt diese Abfrage sehr häufig im Programm auf, so wird man daraus einen Makro machen.

```
        .MACRO VGL
        LDA  %1
        CMP  %2
        BCC  $2
        BEQ  $1
        BCS  $2
$1      LDA  %3
        CMP  %4
$2      NOP
        .ENDM
```

%1, %2, %3 und %4 sind Definitionsnamen, deren eigentlicher Wert bei Aufruf des Makros bestimmt wird.

$1 und $2 sind lokale Sprungmarken. Diese sind nur innerhalb eines Makroaufrufes bekannt.
Muß im Assemblertext dieser Vergleich durchgeführt werden, so wird der Makro durch

```
.
.
STA LISTE
VGL WERTH, ENDH, WERTL, ENDL
BCC WEITER
.
.
```

aufgerufen. An dieser Stelle wird der definierte Text eingefügt und die in der Parameterliste angegebenen Namen den Definitionsnamen der Reihe nach zugewiesen. Innerhalb einer Makrodefinition kann ein anderer Makro aufgerufen, aber kein neuer definiert werden.

Dies wird in dem Programm MAKRODEMO in den Abbildungen 8.2 und 8.3 gezeigt.
Abbildung 8.2 zeigt den Text des Beispiels, wie er mit dem Texteditor des PASCAL-Systems geschrieben wurde. Es gilt hier die Vereinbarung, daß Marken immer an der ersten Stelle der Textzeile beginnen müssen. Bei Assemblerbefehlen muß die erste Zeile immer ein Leerzeichen sein.

```
        .MACRO LOP
        LDY #%1
        LDX #%2
$1      DEY
        BNE $1
        DEX
        BNE $1
        .ENDM
        ;
        .MACRO LUP
        LDA #%1
        STA H
$1      LDA #%2
        STA H1
$2      LOP %3,%4
        STA SPKR
        DEC H1
        BNE $2
```

```
        DEC H
        BNE $1
        .ENDM
        ;
        .PROC TON
SPKR    .EQU 0C030
H       .EQU 10
H1      .EQU 11
        ;
        LUP 10,40,80,01
        LUP 10,40,0FF,01
        ;
        .END
```

8.2 Text Makrodemo

In dem Programm ist in dem Makro LOP eine Zeitschleife programmiert. Als Sprungmarke wird eine lokale Sprungmarke benutzt. Diese beginnt mit einem $-Zeichen, darauf können bis zu 8 Zahlen folgen. Eine lokale Marke ist nur innerhalb eines Makros bekannt. Würde man hier globale Marken (z. B. L oder L20) definieren, würde dies beim Assemblieren dann zu einem Fehler führen, wenn der Makro ein zweites Mal aufgerufen wird.

Der 2. Makro LUP ist ebenfalls eine Zeitschleife. In diesem wird der Makro LOP aufgerufen. Jedesmal wenn dieser verlassen wird, wird am Lautsprecher des APPLE II "gezupft" (LDA SPKR) und somit ein Ton mit einer Tonhöhe erzeugt, die durch die Übergabeparameter bestimmt ist. Die Übergabeparameter für den LUP-Makro bestimmen die Tonlänge.

In dem Programm .PROC TON wird nach den Vereinbarungen nur der Makro LUP mit 4 Parametern aufgerufen. Die beiden ersten werden an die Variablen %1 und %2 übergeben, die beiden anderen an %3 und %4. Dies sind aber die Übergabeparameter für den Makro LOP und dessen Variable %1 und %2.

Was der Assembler aus diesen Makroaufrufen macht zeigt Abbildung 8.3. Hier werden die Übergabeparameter an den richtigen Stellen eingefügt und das vollständige Programm ausgedruckt.

```
PAGE -   1 TON      FILE:SYSTEM.WRK.TEXT

0000ö                                 .PROC TON
Current memory available:     8018
0000ö C030                SPKR        .EQU 0C030
0000ö 0010                H           .EQU 10
0000ö 0011                H1          .EQU 11
0000ö                                 ;
0000ö                                 LUP 10,40,80,01
0000ö A9 10               #           LDA #10
0002ö 85 10               #           STA H
0004ö A9 40               #$1         LDA #40
0006ö 85 11               #           STA H1
0008ö                     #$2         LOP 80,01
0008ö A0 80               #           LDY #80
000Aö A2 01               #           LDX #01
000Cö 88                  #$1         DEY
000Dö D0FD                #           BNE $1
000Fö CA                  #           DEX
0010ö D0FA                #           BNE $1
0012ö 8D 30C0             #           STA SPKR
0015ö C6 11               #           DEC H1
0017ö D0EF                #           BNE $2
0019ö C6 10               #           DEC H
001Bö D0E7                #           BNE $1
001Dö                                 LUP 10,40,0FF,01
001Dö A9 10               #           LDA #10
001Fö 85 10               #           STA H
0021ö A9 40               #$1         LDA #40
0023ö 85 11               #           STA H1
0025ö                     #$2         LOP 0FF,01
0025ö A0 FF               #           LDY #0FF
0027ö A2 01               #           LDX #01
0029ö 88                  #$1         DEY
002Aö D0FD                #           BNE $1
002Cö CA                  #           DEX
002Dö D0FA                #           BNE $1
002Fö 8D 30C0             #           STA SPKR
0032ö C6 11               #           DEC H1
0034ö D0EF                #           BNE $2
0036ö C6 10               #           DEC H
0038ö D0E7                #           BNE $1
003Aö                                 ;
003Aö                                 .END
```

8.3 Assemblertext Makrodemo

Die Verwendung von Makros ist dann sinnvoll, wenn immer wiederkehrender Text in einem Assemblerprogramm auftritt.

Bei der Assemblierung können Teile des Quelltextes durch eine IF . . ELSE-Anweisung davon ausgenommen werden.

Beispiel:

```
.IF ⟨ AUSDRUCK ⟩
⟨ TEXT1 ⟩
[ . ELSE ]
⟨ TEXT2 ⟩
.ENDC
```

Der TEXT1 wird nur dann assembliert, wenn ⟨ AUSDRUCK ⟩ ungleich Null ist. Dann aber wird TEXT2 (wenn vorhanden) übersetzt.

Folgende Befehle werden im PASCAL-Assembler abweichend von anderen 6502 Assemblern geschrieben:

```
LDA @ ADR,Y  =  LDA (ADR),Y
LDA @ ADR,X  =  LDA (ADR,X)
JMP @ ADR    =  JMP (ADR)
```

8.13 Die wichtigsten Monitorroutinen

OUTCHR   Ausgabe eines Zeichens auf den Bildschirm.
Adresse: $FDED

Dieses Unterprogramm gibt ein Zeichen an der augenblicklichen Cursorposition aus. Er springt dabei über einen Vektor in den Zellen $36 und $37 in die eigentliche Ausgaberoutine COUT1 bei Adresse $FDF0. Dadurch kann man leicht die Ausgabe auf ein anderes Ausgabegerät umsteuern. Der Vektor zeigt auf den Beginn des eigenen Ausgabeprogramms. An dessen Ende springt man mit einem JMP $F0F0 in die Bildschirmausgabe.
Die Registerinhalte bleiben erhalten.

PRTBYT   Adresse $FDDA

PRTBYT gibt den Akkumulatorinhalt in Form von 2 Hexadezimalzeichen auf den Bildschirm aus. Der Akkumulatorinhalt wird zerstört.

RDKEY   Eingabe eines Zeichens vom Tastenfeld
Adresse $FD0C

Dieses Unterprogramm springt über einen Vektor in den Zellen $38 und $39 in das Programm KEYIN bei Adresse $FD1B.
Wie bei der Ausgabe kann durch Umsteuern des Vektors, die Eingabe eines Zeichens von einem anderen Gerät durch ein eigenes Programm erfolgen.

8.2 ATARI

8.21 ATARI – Editor/Assembler für ATARI 400 und ATARI 800

Der ATARI-Assebler besteht aus 3 Teilen. Dem Text-Editor, dem eigentlichen Assembler und einem "Debugger", ein Programm zum Testen von Maschinenprogrammen. Wir wollen uns hier nur mit den ersten beiden Teilen befassen.

Der Text-Editor ist zeilenorientiert. Die Zeilennumerierung erfolgt automatisch nach Eingabe des Befehls NUM. Ohne weitere Optionen wird die Zeilennumerierung mit 10 begonnen und jeweils um 10 erhöht. Folgenden Optionen sind möglich:

NUMnn	Die Zeilennumerierung wird um nn erhöht.
NUMmm,nn	Die Zeilennumerierung beginnt bei mm und wird um nn erhöht. Die vorgegebene Numerierung wird durch Eingabe von 2 mal RETURN ausgeschaltet.

Die Löschung einer Zeile erfolgt durch DEL.

DELnn	Löschung der Zeile nn.
DELnn,mm	Löschung der Zeilen nn bis mm.
NEW	Löscht den Textspeicher.

Mit REN lassen sich die Zeilen neu numerieren. Dabei gelten die gleichen Optionen wie bei NUM.

REN	Zeilennumerierung von 10 in 10-er Schitten.
RENnn	Zeilennumerierung in Schritten von nn.
RENmm,nn	Zeilennumerierung beginnt bei mm und wird in Schritten von nn erhöht.

Neben diesen Anweisungen besitzt der Texteditor noch die Befehle FIND zum Suchen einer Zeichenkette und REP (Replace) zum Suchen und Ändern einer Zeichenkette im Textspeicher.

Der Assembler kennt folgende Pseudo-Befehle:

Die * - Anweisung.
Beispiel: 10 * = $6000
Damit wird die Anfangsadresse des übersetzten Programms festgelegt. Falls keine neue Festsetzung folgt, beziehen sich alle folgenden Adressen auf diese Anfangsadresse.

Die .BYTE - Anweisung
Beispiel: 20 .BYTE 22
Damit wird im Programm ein Byte Speicherplatz mit der Dezimalzahl 22 belegt. Folgen mehrere Zahlen, getrennt durch Komma, so belegen diese die folgenden Byts.
Beispiel: 20 .BYTE 22,33,21

Soll an dieser Stelle ein Text im Programm gespeichert werden, so ist dieser Text in Anführungszeichen zu setzen.
Beispiel: 20 .BYTE "DIES IST TEXT"

Man beachte, Daß Pseudo-Befehle, die aus einem Wort bestehen, mit einem Punkt (.) beginnen.

Die .DBYTE - Anweisung
Diese Anweisung reserviert 2 Byte Speicherplatz.
Beispiel: 40 .DBYTE $1F26
Dabei wird die Zahl $1F26 in dieser Reihenfolge gespeichert. Ist die augenblickliche Adresse $4010 so wird in $4010 $1F und in $4011 $26 gespeichert.

Die .WORD - Anweisung
Diese Anweisung entspricht der .DBYTE Anweisung. Es werden wiederum 2 Speicherplätze reserviert, nur wird die Zahl in umgekehrter Reihenfolge eingeschrieben.
Beispiel: 40 .WORD $1F26
Wenn wir wieder bei der Adresse $4010 sind, so wird in Adresse $4010 $26 und in $4011 $1F gespeichert. Da bei 6502 Systemen auf diese Weise Adressen gespeichert werden, wird man mit der .WORD Anweisung Adresstabellen anlegen.

Die .END - Anweisung
Mit dieser Anweisung wird der duelltext abgeschlossen. Sie muß am Ende jedes Programms auftreten.

Die = - Anweisung
Mit dem Gleichheitszeichen werden Marken im Programm Adressen zugewiesen.
Beispiel: 10 MARKE = $1000
Der symbolische Name MARKE entspricht der Adresse $1000.
In dieser Zeile darf nach der Zeilennummer nur **ein** Leerzeichen folgen.
Die Adressenzuweisung an Marken kann auch durch Adressrechnung erfolgen:
Beispiel: 10 * = $1000
20 TAB = * + $20
TAB entspricht dann in diesem Beispiel der Adresse $1020.

Weitere Pseudo-Befehle sind:

10 .TITEL "NAME"	Eingabe eines Programmnamens.
20 .PAGE "TEXT"	Eingabe einer Seitenüberschrift.
40 .TAB n1,n2,n3	Tabellierung des Ausdrucks, wobei im Ausdruck des übersetzten Programms das Operationscodefeld bei n1, das Adressfeld bei n2 und das Kommentarfeld bei n3 beginnt.

Mit 4 .OPT - Anweisungen kann die Assemblierung beeinflusst werden.

100 .OPT NOLIST 200 .OPT LIST	Der Text zwischen Zeile 100 und 200 wird bei der Übersetzung nicht ausgegeben.
100 .OPT NOOBJ 200 .OPT OBJ	Der Text zwischen den Zeilen 100 und 200 wird nicht übersetzt, der Speicherplatz aber freigehalten.
100 .OPT NOERR 200 .OPT ERR	Die Ausgabe von Fehlern, die zwischen den Zeilen 100 und 200 bei der Übersetzung auftreten, werden nicht ausgegeben.

100 . OPT NOEJECT Kein zusätzlicher Zeilenvorschub bei einen
200 .OPT EJECT Seitenvorschub zwischen den Zeilen 100 und 200.

Mit der IF-Anweisung können Teile des Quellcodes von der Übersetzung ausgenommen werden.

Beispiel:
```
 50 Z=0
100 .IF Z @MARKE
 .
 .
 .
150 MARKE
```

Der Quelltext in den Zeilen 100 bis 150 wird nur übersetzt, wenn Z Null ist. Ändert man Zeile 50 in Z=1 ab, so wird der Text in den Zeilen 100 bis 150 nicht übersetzt, und auch kein Speicherplatz freigehalten.

Die Eingabe von Kommentar erfolgt nach der Adresseneingabe. Beide müssen durch ein Leerzeichen getrennt sein. Als Trennzeichen kann auch der Stichpunkt (;) verwendet werden. Soll eine Kommentarzeile im Text eingefügt werden, so beginnt diese mit einem Strichpunkt.

Allgemein ist beim Schreiben eines Programmes folgendes zu beachten: Zwischen der Numerierung und einer Marke muß **ein** Leerzeichen eingefügt sein, zwischen Numerierung und einem Assembler oder Pseudobefehl müssen **zwei** Leerzeichen vorhanden sein. Eine Fehlermeldung wird erst bei der Assemblierung ausgegeben.

Die Schiebebefehle, die sich auf den Akkumulator beziehen lauten hier:

LSR A
ASL A
ROL A
ROR A.

Beim Assemblieren mit dem ATARI ist immer der Quelltext und das übersetzte Programm im Speicher vorhanden. Man muß deshalb dafür sorgen, daß sich diese beiden Programme nicht überschreiben und daß sie auch nicht Speicherbereiche des Monitors und des Bildspeichers benutzen.

Mit der Monitoranweisung SIZE werden drei Adressen ausgegeben.

Beispiel: SIZE
Rechner antwortet: 0700 0880 5C1F
$700 ist die Anfangsadresse, $880 die Endadresse des Textspeichers. Von hier bis zur Adresse $5C1F ist der Speicher frei für Maschinencode.

Schreiben eines Programms.

Nach Einstecken des Moduls Assembler befindet sich der Rechner im Texteditor. Der Quelltext kann nun geschrieben werden, wobei zu beachten ist, daß Marken nur ein Leerzeichen zwischen Zeilenzahl und ersten Buchstaben enthalten, bei Assemblerbefehlen aber zwei Leerzeichen vorhanden sein müssen. Der Quellcode wird mit .END abgeschlossen. Die Übersetzung wird durch den Befehl ASM eingeleitet. Ohne weitere Optionen wird der Quellcode aus dem Textpuffer übersetzt, der Ausdruck erfolgt auf dem Bildschirm, und das Maschinenprogramm wird bei der angegebenen Anfangsadresse in das RAM abgelegt. Soll die Ausgabe auf den Drucker erfolgen, so ist die Option #P notwendig. Der Befehl lautet dann:

ASM,#P

Durch die Eingabe von LIST erfolgt die Ausgabe des Textpuffers auf den Bildschirm. Es findet keine Übersetzung statt. Bei LIST können folgende Optionen angegeben werden:

LIST#E (gleich wie LIST)	Bildschirmausgabe
LIST#P	Druckerausgabe
LIST#C	Speicherung auf Kassette.

(Anmerkung: Hier kein Komma zwischen Befehl und Option)

Weitere Befehle sind:

PRINT	Drucken ohne Zeilennummer.
ENTER	Einlesen eines Quellprogramms.
SAVE	Speichern eines Maschinenprogramms.

LOAD	Laden eines Maschinenprogramms.

8.22 Monitor und Speicherbelegung

Der Monitor des ATARI-Computers weist gegenüber anderen Monitoren einige Besonderheiten auf. Er besteht aus 3 Teilen:

1.	CIO	Programmteil für allgemeinen I/O Datentransfer
2.	SIO	I/O-Programmteil für serielle Datenübertragung
3.	IH	Interrupt - Service - Routinen

Die Verwendung von Interrupts bedingt gleichfalls die Verwendung von Tabellen für den augenblicklichen Maschinenzustand und Sprungtabellen für Programmverzweigungen.

Für den Programmablauf der CIO-Routine sorgt eine in den Zellen $340 bis $348 gespeicherte Tabelle IOCB (Input Output Control Table) für die Ausführung eines Input-Output Datentransfers. Vom Benutzer kann dabei der Befehl ($342), die Puffer-Adresse ($344, $345), die Pufferlänge ($348, $349) und eine zusätzliche Information ($34A, $34B) übergeben werden.

Beispiel: Tabelle für die Bildschirminitialisierung nach dem Einschalten des Rechners:

$342	$03	Befehl OPEN
$344	$18	F118 = Adresse des Textes E : EOL
$345	$F1	($45, $3A, $9B)
$34A	$0C	Bitmuster für "Open for all Devices, Input or Output"

Mit dieser Tabelle und einem JSR CIO wird der Bildschirm initialisiert.

Verfolgt man die "Power up" Routine, so findet man hier die Bedeutung einiger Speicherzellen.

Die Speicherzelle $52 ist der linke Bildschirmrand, in $53 der rechte

Bildschirmrand angegeben.
Bei der Initialisierung enthalten die Zellen $02 und $27.

Weiter wird festgestellt, ob ROM-Kassetten oder RAM in den Steckplätzen A und B vorhanden sind. Dabei wird zuerst Steckplatz B untersucht. Ist der Inhalt der Adresse $9FFC Null, und lässt dieser sich nicht ändern, so ist eine ROM-Kassette vorhanden und das Programm dieser Kassette wird durch einen indirekten Sprung über die Adresse $9FFE gestartet.

Ist keine Kassette oder RAM an diesem Steckplatz so wird der andere Platz A untersucht. Die Adressen dieses Steckplatzes sind $BFFC und $BFFE. Das Vorhandensein einer ROM-Kassette wird in den Speicherzellen $06 für A-Kassette und $07 für B-Kassette mit einer 1 vermerkt. Nach Abschluß des Initialisierungsprogramms wird, wenn keine ROM-Kassette, keine Disk und keine Kassette gebootet werden konnte, das Blackboard-Programm über den DOS-Vektor in den Speicherzellen $0A und $0B angesprungen.

In Abbildung 8.2 ist die Speicherbelegung des ATARI 800 dargestellt.

Es folgen nun die wichtigsten Monitor-Unterprogramme.

8.23 Einige Monitorunterprogramme

Unterprogramm KGETCH
Adresse $F6DD
Dieses Programm übernimmt ein Zeichen vom Tastenfeld. Beim Rücksprung ist dieses Zeichen im Akkumulator und kann von dort ins Hauptprogramm übernommen werden. X und Y Register werden verändert.

Unterprogramm EGETCH
Adresse $F63E
Dieses Programm übernimmt ein Zeichen vom Tastenfeld und gibt es bei der augenblicklichen Cursorposition aus. Ein Rücksprung aus dieser Routine erfolgt durch RETURN, dabei ist das zuerst eingegebene Zeichen im Akkumulator.

Adresse (dez.)	Adresse (hex)	Belegung
65535	$FFFF	
		MONITOR ROM
57344	$E000	
		GLEITKOMMA ROM
55296	$D800	
		HARDWARE REG. (D400 – D41F ANTIC; D300 – D31F PIA; D200 – D21F POKEY; D000 – D01F CTIA)
53248	$D000	
		NICHT BELEGT
49152	$C000	
		ROM KASSETTE A ODER RAM
40960	$A000	
		ROM KASSETTE B ODER RAM
32768	$8000	
		BILDSPEICHER TEXT (MODE 0)
31755	$71CF	
		FREIES RAM
10880	$2A80	
		BASIC, DOS BUFFER
9856	$2680	
		DOS
4864	$1300	
		SYSTEM RAM DOS
1792	$700	
		FREIES RAM
1536	$600	
		GLEITKOMMA RAM
1406	$57E	
		ROM – KASSETTE BASIC RAM
1152	$480	
		MONITOR RAM
512	$200	
		STAPPELSPEICHER
256	$100	
		FP–RAM BASIC–RAM ($B0 – $CF FREI)
128	$80	
		ZERO PAGE RAM MONITOR
0	$00	

8.2 **Speicherbelegung ATARI**

Dieses Programm wird in der BLACKBOARD-Routine verwendet.

```
ANF   JSR EGETCH
      CLC
      BCC ANF
```

Es kann nur durch BREAK oder durch Ausschalten des Rechners verlassen werden.

Unterprogramm EOUTCH
Adresse $F6A4

Dieses Programm gibt das ATASCII-Zeichen im Akkumulator bei der augenblicklichen Cursorposition aus. Nach dem Rücksprung ist es noch im Akkumulator vorhanden, X- und Y-Register sind verändert.

Für die Cursorsteuerung stehen folgende Sonderzeichen zur Verfügung. Eine Ausgabe dieser Zeichen mit EOUTCH bewirkt folgende Funktion:

$70	Löscht den Bildschirm und setzt den Cursor in die linke obere Ecke.
$1C	Der Cursor wird eine Zeile nach oben gesetzt.
$1D	Der Cursor wird eine Zeile nach unten gesetzt.
$1E	Der Cursor wird eine Stelle nach links gesetzt.
$1F	Der Cursor wird eine Stelle nach rechts gesetzt.

In allen Fällen wird die Cursorpositionierung am anderen Bildrand weitergeführt, wenn ein Bildrand überschritten worden ist.

$FF	Dieses Steuerzeichen schiebt den Text rechts vom augenblicklichen Cursorstand eine Stelle nach rechts und macht somit Platz zum Einfügen eines Zeichens.
$90	Alle Zeilen, von der augenblicklichen Cursorposition an und darunter werden eine Zeile tiefer gesetzt und eine Leerzeile eingefügt. Der Cursor steht am Anfang dieser Leerzeile.

$7E Der Cursor wird eine Stelle nach links gesetzt, aber nicht über den Anfang der Zeile hinaus.

$9C Löscht die Zeile mit der augenblicklichen Cursorposition. Alle darunterliegenden Zeilen werden hochgesetzt.

$FE Löscht den Buchstaben bei der augenblicklichen Cursorposition. Alle Zeichen rechts davon werden eine Stelle links geschoben.

$FD Gibt einen Ton aus, der Text wird nicht verändert.

$9B Dieses Zeichen bedeutet EOL (End Of Line) und wird als Begrenzung einer Zeile oder eines Datensatzes verwendet. Seine Funktion entspricht dem nicht vorhandenen ASCII-Zeichen RETURN.

Unterprogramm PUTLIN
Adresse $F385

Dieses Unterprogramm gibt einen im Rechner gespeicherten Text aus. Die Anfangsadresse des Textes wird im X- und Y-Register übergeben, mit dem LSB der Adresse im X-Register und dem MSB der Adresse im Y-Register. Als Textendezeichen wird $9B verwendet. Ist kein Textendezeichen vorhanden, so werden 255 Byte, als ATASCI-Zeichen interpretiert und ausgegeben.

Das folgende Programm gibt einen Text in der obersten Bildzeile aus:

```
  2 EOUTCH = $F6A4
  4 PUTLIN = $F385
 10    * = $0600
 20    LDA #$70            Bildschirm löschen
 30    JSR EOUTCH
 40    LDX #50             Anfangsadresse setzen
 50    LDY #06
 60    JSR PUTLIN          Text ausgeben,
 70    BRK                 Programm abbrechen.
 80;
100    * = $0650
110    .BYTE "DIESER TEXT WIRD AUSGEGEBEN"
120    .BYTE $9B
999    .END
```

Ein weiteres Beispiel ist ein Programm für einen Speicherauszug (HEXDUMP).
Es benutzt das schon in Kapitel 7.7 beschriebene Unterprogramm PRTBYT.

```
00B0          10 STARTL     =   $B0
00B2          20 ENDEL      =   $B2
00B4          30 COUNT      =   $B4
F6A4          40 OUTCH      =   $F6A4
0000          45            *=  $2000
2000 A000     50 DUMP       LDY #$00
2002 84B4     60            STY COUNT
2004 A000     63 M0         LDY #$00
2006 B1B0     65            LDA (STARTL),Y
2008 203520   70            JSR PRTBYT
200B E6B4     80            INC COUNT
200D A5B4     90            LDA COUNT
200F C908     0100          CMP #08
2011 D009     0110          BNE M1
2013 A99B     0120          LDA #$9B
2015 20A4F6   0130          JSR OUTCH
2018 A900     0140          LDA #$00
201A 85B4     0150          STA COUNT
201C E6B0     0160 M1       INC STARTL
201E D002     0170          BNE M
2020 E6B1     0180          INC STARTL+1
2022 A5B1     0190 M        LDA STARTL+1
2024 C5B3     0200          CMP ENDEL+1
2026 90DC     0210          BCC M0
2028 F002     0220          BEQ M2
202A B008     0230          BCS DRTS
202C A5B0     0240 M2       LDA STARTL
202E C5B2     0250          CMP ENDEL
2030 90D2     0260          BCC M0
2032 F0D0     0270          BEQ M0
2034 60       0280 DRTS     RTS
2035 85B5     0290 PRTBYT   STA COUNT+1
2037 4A       0300          LSR A
2038 4A       0310          LSR A
2039 4A       0320          LSR A
203A 4A       0330          LSR A
203B 204B20   0340          JSR HEXTA
```

```
203E A5B5   0350        LDA  COUNT+1
2040 204B20 0360        JSR  HEXTA
2043 A920   0364        LDA  #$20
2045 20A4F6 0366        JSR  OUTCH
2048 A5B5   0370        LDA  COUNT+1
204A 60     0380        RTS
204B 290F   0390 HEXTA  AND  #$0F
204D C90A   0400        CMP  #$0A
204F 18     0410        CLC
2050 3002   0420        BMI  HEXTA1
2052 6907   0430        ADC  #$07
2054 6930   0440 HEXTA1 ADC  #$30
2056 4CA4F6 0450        JMP  OUTCH
2059        0460        .END
```

8.21 Programm Hexdump-ATARI

Die für dieses Programm benötigten Speicherzellen in der Zero-Page wurden auf die Adressen $B0 – $B5 gelegt. Die Plätze von $B0 – $CF ist der für den Benutzer freie Speicherbereich, die Assembler, Gleitkomma-Unterprogramme und BASIC nicht verwenden.

Der Monitor benötigt ausnehmend viele Zellen in der Zero-Page ($00 – $7F). Diese enthalten hauptsächlich Merker und Adressen.

Die Programmierung der Farben, Töne und der Grafik erfolgt über Hardware-Register ($D000 – $D41F). Das Umschalten eines Bytes ändert z. B. augenblicklich die Hintergrundfarbe.

Hier bietet der ATARI dem Programmierer in Maschinencode ausgezeichnete Möglichkeiten.

8.3 Programmieren in Maschinensprache mit Commodore Rechnern

Die Programmierung der PET/CBM/VIC Rechner ist vom Prinzip her sehr einfach, wenn Sie dieses Buch bis hierher studiert haben. Probleme gibt es dabei nur durch den ständigen Wechsel der Betriebssysteme, Zeropagebelegungen. Es gibt jedoch in den CBM- und Pet-Geräten mit den verschiedensten Betriebssystemen oben im Speicher (nähe FFFF) eine Sprungtabelle mit den wichtigsten Routinen für den Maschinensprachenprogrammierer.



PET 2001 (alte ROMs)

In den neuen CBM-Systemen 3001 und 4001 ist ein Monitor eingebaut. Er kann mit SYS(1024) aufgerufen werden. Für die älteren PETs gibt es einen Monitor auf Cassette.

Er erlaubt dem Anwender Speicherzellen anzusehen und zu ändern:

M 033A, 03EE

bringt den Inhalt der Zellen 033A – 03EE Hex auf den Bildschirm. Sie können dann mit dem Cursor diese Zeilen abändern oder z. B. ein Programm eingeben.

Mit G XXXX können Sie ein Maschinenprogramm an Adresse XXXX Hex starten.

Der Befehl X erlaubt Ihnen die Rückkehr nach BASIC, L und S ermöglichen ein Laden und Speichern von Maschinenprogrammen auf Cassettenband (Siehe hierzu PET-User Manual)



Der VC-20 Volkscomputer mit Cassettenrecorder

Die wichtigsten Adressen in der Sprungtabelle für unsere kleinen Versuche sind:

Funktion	Adresse in der Sprungtabelle
Eingabe eines Zeiches von der Tastatur (aus Puffer) GET a. Ch.	FFE4
Eingabe eines Zeichens (RDT)	FFCF
Ausgabe eines Zeichens auf dem Bildschirm	FFD2
Bildschirmspeicher	ab 8000 Hex = 32768 Dezimal
ASCII nach HEX-Wandlung und Ablage im Akku (HEXIT)	0685
Wagenrücklauf CRLF	04F2
Rücksprung Monitor	0400
LOAD Routine	FFD5
SAVE Routine	FFD8
STOP Cassettenmotor	FFED – FFFA
RESET	FFFC – FFFD
NMI	FFFA – FFFB
IRQ	FFFE – FFFF
Cassettenpuffer für Cass. II	826 – 1017 Dez 033A – 03F9 Hex
Bildschirm löschen SYS(57910) beim PET 2001 (alte ROMs)	E236 Hex
Bildschirmlöschen bei der 3000er Serie SYS(57829)	E1E5 Hex

Freie "Zero-Page"-Adressen $23 – $5A Hex, wenn Sie im Monitor programmieren.

Wir wollen jetzt unser Standardprogramm einmal auf dem CBM 3001 implementieren:

Eingabe eines Zeichens von der Tastatur auf den Bildschirm:

Wir legen unser Programm in einen Adressbereich des CBM, der auch in BASIC geschützt ist, nämlich ab Adresse 033A (zweiter Cassettenpuffer).

```
033A    JSR    $FFCF    20CFFF
033D    JSR    $FFD2    20D2FF
0340    JMP    $033A    4C3A03
```

Geben Sie dieses Programm mit Hilfe des Monitorbefehles M ein und starten Sie mit G 033A.

Geben Sie Werte von der Tastatur ein und beobachten Sie den Bildschirm. Da der CBM keine BREAK-Taste hat, muß das Gerät ausgeschaltet, und alles wieder neu eingegeben werden.

Wir wollen jetzt eine Abfrage einbauen, die uns ermöglicht, durch Drücken der A-Taste unser kleines Programm wieder zu verlassen.

Nach Eingabe des Zeichens von der Tastatur her, fragen wir nach dem Hex-Zeichen 41 = ASCII A ab.
Dies geschieht durch die Befehle CMP und BEQ.

```
033A    20E4FF
033D    C941
033F    F006
0341    20D2FF
0344    4C3A03
0347    4C0004
```

Wir verwenden diesmal die GET Routine, welche das Zeichen von der Tastatur unmittelbar in den Akkumulator übernimmt.

Wie der CMP und BEQ-Befehl arbeiten haben wir bereits kennengelernt.

Ein weiteres Beispiel in Maschinensprache

Ausdruck des Zeichensatzes auf den Bildschirm:

```
LDX      #$00
TXA LP   TXA
STA      $8150,X
INX
BNE LP
JMP      $0400
```

In Hexadezimal übersetzt ergibt sich:

```
033A   A4   00   8A   9D   50   81   E8   D0
0342   F9   4C   00   04   00
```

Dieses Maschinenprogramm können Sie auch von BASIC über SYS(826) aufrufen (33A Hex = 826 Dezimal). Vorher müssten Sie jedoch den Rücksprung in den Monitor in Zelle 0343 Hex in RTS = 60 Hex abwandeln.

Hier eine kurze Beschreibung dieses kleinen Maschinensprachenprogrammes:

Schritt	Text
1.	Lade das X-Register mit 0
2.	Bringe den Inhalt des X-Registers in den Akkumulator
3.	Lade den Inhalt des Akkus in Zelle $8150 (Zelle 8150 ist die Postleitzahl von Holzkirchen und die 8 Zeile auf dem Bildschirm des CBM (PET)).
4.	Addiere eine 1 zum Inhalt des X-Registers hinzu
5.	Springe nach Schritt 2 solange, bis X nicht Null ist. (Das X-Register wird null, wenn es bis FF = 256 hochgezählt wurde). Dadurch erhalten wir eine Schleife bis 256.
6.	Rücksprung in Monitor JMP 0400 oder RTS, wenn von BASIC aufgerufen.

Ein äquivalentes BASIC-Programm würde wiefolgt aussehen:

```
 5 GOSUB 10
 7 END
10 I=0
20 POKE 33104 + I,I
30 LET I = I + 1
40 IF I < 256 GOTO 20
50 RETURN
```

Programmieren in Maschinensprache mit VC-20 Volkscomputer von Commodore

Die Programmierung des Volkscomputers VC-20 von Commodore erfolgt ähnlich wie die der PET/CBM-Serie.
Die Kernel Einsprungadressen sind gleich geblieben. Wir haben folgende Routinen im Kernel identifizieren können:

FFD2 Hex für CHROUT =	Ausgabe eines Zeichens auf dem Bildschirm
FFE4 Hex für GETIN =	Eingabe eines Zeichens von der Tastatur.
FFCF Hex für CHRIN =	Eingabe eines Zeichens
FFC6 Hex für CHKIN =	Device-Bestimmung von wo aus eingegeben werden kann.

Geändert hat sich der verfügbare Speicherbereich. In der Grundversion beginnt der RAM-Bereich für Anwenderprogramme in BASIC an Adresse 1000 Hex = 4096 Dezimal. Der Bereich von 400 Hex – FFF Hex ist in der Grundversion nicht bestückt (ähnlich wie KIM-1 und SYM).

Der Bildschirmspeicher liegt nicht wie beim PET/CBM ab Adresse 8000 Hex – 8400 Hex sondern ab Adresse 1E00 – 1FFF Hex = 7680 – 8191 Dezimal.

Achtung !
Der VC-20 hat nur noch einen Cassettenpuffer ab Adresse 033C Hex = 828 Dezimal bis 1023 Dezimal. Diesen können Sie jetzt nicht mehr so problemlos für Anwenderprogramme verwenden, wenn Sie später auf

Cassette speichern wollen.

POKER MAKER

```
500 PRINT"3ML TO BASIC PROGRAMMER"
502 PRINT:PRINT"THIS PROGRAM GENERATES A BASIC P
ROGRAM"
504 PRINT"WHICH LOADS A MACHINE LANGUAGE ROUTINE
":PRINT
506 REM DIESES PROGRAMM ERZEUGT EIN MASCHINEN-
507 REM PROGRAMM IN DATA ZEILEN.WELCHE
508 REM DANN EINGEPOKED WERDEN KOENNEN.
510 REM ACHTEN SIE DARAUF,DASS SIE SINN-
511 REM VOLLEN CODE IN DEN RAM ZELLEN HABEN.
512 INPUT "STARTING LOCATION";L1
514 INPUT "ENDING LOCATION";L2
516 IF L2<=L1 THEN 500
518 PRINT "3"
520 PRINT "20FORI=";L1;"TO";L2;":READDC:POKEI,DC
:NEXTI"
522 PRINT "L1=";L1;":L2=";L2;":GOTO528"
524 POKE 525,10:FOR N=0 TO 9:POKE 527+N,13:NEXT
N
526 PRINT "":END
528 ML=L1:LN=22
530 PRINT "3"
532 PRINT LN;"DATA";
534 D1=PEEK(ML):D1$=STR$(D1)
536 DC$=RIGHT$(D1$,LEN(D1$)-1)
538 PRINT DC$;:ML=ML+1
540 IF POS(0)<36 THEN 544
542 PRINT:PRINT "LN=";LN;"+2:ML=";ML;":L2=";L2;"
:GOTO530":GOTO 548
544 IF ML>L2 THEN PRINT:PRINT "GOTO554":GOTO 548
546 PRINT CHR$(44);:GOTO 534
548 PRINT ""
550 POKE 525,10:FOR N=0 TO 9:POKE 527+N,13:NEXT
N
552 PRINT "":END
554 I=500:J=502:K=504
556 REM
```

```
558 REM
560 PRINT "3":PRINT I:PRINT J:PRINT K
562 PRINT "I=";I+6;":J=";J+6;":K=";K+6;":GOTO560
"
564 PRINT "":POKE 525,10:FOR N=0 TO 9:POKE 527+N
,13:NEXT N
READY.
```

Dieses Programm läuft nur auf den alten PET's mit ROM-Version 1. Für CBM-Systeme der 3000er oder 4000er Serie müsste er erst abgewandelt werden. POKER MAKER generiert ein Programm, welches Dezimalwerte in den Speicherbereich bringt. Geben Sie Anfangs- und Endaddresse in dezimaler Form ein. Nach einigen Sekunden steht Ihr gewünschtes Programm im Speicher.

Besitzer, welche einen CBM der 3000er Serie haben, können das Programm für sich wiefolgt abändern.

In den Zeilen 524, 550 und 564 muß die Zahl 525 in 158 und 527 in 623 abgewandelt werden. Dann läuft der POKER MAKER auch auf CBM.

Die Routine arbeitet in dieser geänderten Version auch auf der 4000er Serie.

Es empfiehlt sich daher Maschinenprogramme nach 1000 Hex Anfangsadresse zu legen.

Ein weiterer Bereich, der von Maschinensprachenprogrammierer noch genutzt werden kann ist der BASIC-Arbeitsbereich ab Adresse 0000 Hex – 0090 Hex (0 – 144 Dezimal). BASIC im ROM befindet sich an der gleichen Stelle im Speicher wie bei den CBM/PET-Maschinen, nämlich von C000 – E000 Hex.

Unser kleines Beispielprogramm müsste deshalb für den VC-20 wie-

folgt aussehen:

```
LDX    #$00
TXA LP   TXA
STA    $1E58
INX
BNE LP
RTS
```

Die Anfangsadresse des Bildschirmes wurde entsprechend abgeändert (siehe STA $1E58).

Das Programm sähe dann wiefolgt aus:

1000	A2	00	8A	9D	58	IE	E8	D0
1009	F9	60						

Da der VC-20 in der Grundversion keinen eingebauten Maschinensprachenmonitor besitzt, müssen Sie dieses Programm über BASIC in den Speicher "poken" und mit SYS(4096) von BASIC aus starten.

Hierzu übersetzen wir zuerst die Hexadezimalzahlen in Dezimalzahlen.

Hex:	A2	00	8A	9D	58	IE	E8	D0	F9	60
Dez.:	162,	00,	138,	157,	88,	30,	232,	208,	249,	96

Das zu verwendende BASIC-Programm für Ihren VC-20 muß nun wiefolgt aussehen:

```
 10 FOR X = 4096 TO 4105
 20 READ Y
 30 POKE X,Y
 40 NEXT X
 50 DATA 162,00,138,157,88,30
 60 DATA 232,208,249,96
100 END
```

Geben Sie dieses kleine BASIC-Programm nun in Ihren VC-20 ein und starten Sie es mit RUN. Danach geben Sie SYS(4096) ein und sehen

sich das Ergebnis auf dem Bildschirm an. Achten Sie einmal auf die Geschwindigkeit der Programmausführung. Für größere Programme dieser Art gibt es spezielle Utilities, welche ein Maschinenprogramm im Speicher automatisch in DATA-Zeilen mit entsprechenden Dezimalwerten umwandelt (POKER MAKER).

Das Programm Eingabe eines Zeichens und Darstellung auf dem Bildschirm würde dann wiefolgt aussehen:

```
1000    JSR    $FFCF    20CFFF
1003    JSR    $FFD2    20D2FF
1006    JMP    $1000    4C0010
```

Geben Sie dieses Programm einmal über Ihren Monitor ein. Wenn kein Monitor vorhanden, kann es wieder an Hand der vorher besprochenen Methode in Dezimalzahlen umgerechnet und in den Speicher "gepoked" werden. Der Start von BASIC geschieht wieder mit SYS(4096) Sie müssten eigentlich selbst in der Lage sein, eigene kleine Maschinenprogramme zu schreiben und auf Ihren VC-20 auszutesten.

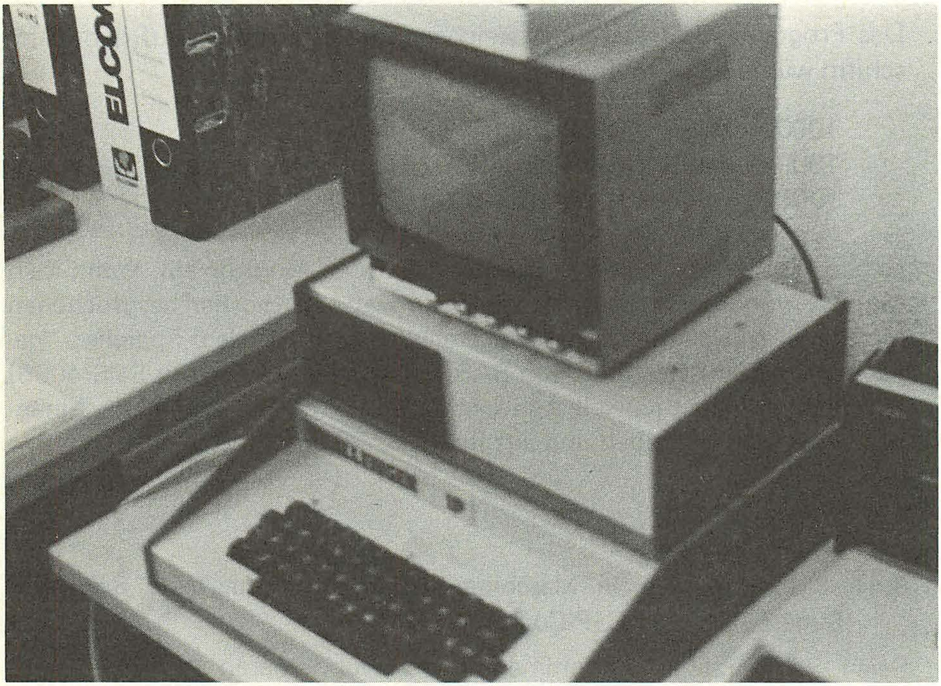
8.4 Programmieren in Maschinensprache mit dem Ohio Scientific Superboard oder C1P-Computer

Seien Sie herzlich willkommen in der wunderbaren Welt der 6502 Maschinensprache auf dem Ohio Superboard. Wir gehen davon aus, daß Sie bereits in BASIC programmiert haben und sich jetzt mit den wichtigsten Grundlagen der 6502 Maschinensprache auf Ihrem System vertraut machen wollen. Die folgende kleine Einführung soll Sie mit den Grundlagen vertraut machen und besonders die systemspezifischen Eigenschaften herausstellen. Weiterhin soll Sie Ihnen als Starthilfe für Ihre weiteren eigenen Maschinensprachen-Experimente dienen.

Soweit wie möglich, wollen wir immer vom Flußdiagramm ausgehen, welches wir zuerst gemeinsam erarbeiten. Wenn das Flußdiagramm erstellt ist, werden wir dann unser Programm im Maschinencode erstellen.

Der Monitor des Superboards (C1P) ist recht dürftig, so daß es sich empfiehlt einen komfortableren Monitor anzuschaffen (Supermonitor/

Extended Monitor etc.). Der "eingebaute" Monitor kann nach dem Einschalten oder nach BREAK über M initialisiert werden. Dieser 65 V Monitor befindet sich in allen Ohio Scientific Computern.



Der Superboard C1P-MF

Er belegt die Speicheradressen ab FE00 Hex bis FEFF. Er verwendet 256 Byte RAM ab Adresse 0100 Hex und 5 Bytes oben an der Grenze der Zero-Page (FB – FF).

Der 65 V-Monitor kennt zwei verschiedene "Modes".

1. Adressmode
2. Data-Mode

Befehle:

/	=	Wechsel in den Data-Mode
GXXXX	=	Start an Adresse XXXX
L	=	Laden von Cassette (Maschinencode)
.	=	Wechsel in den Adressmode
RETURN	=	Öffnen der nächsten Adresse (Erhöhen der Adresse um 1)

Alle Tasten, die keine Kommandos oder nicht Hexadezimalzahlen liefern, werden vom Monitor nicht anerkannt.

Wichtige Adressen und Marken:

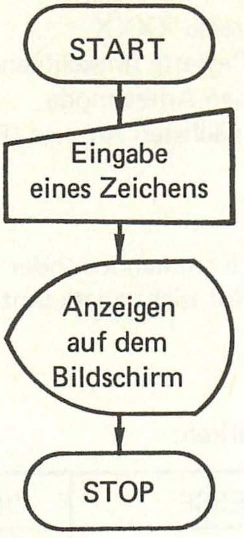
MARKE	ADRESSE	BEZEICHNUNG
VM	FE00	RESTART MONITOR
	FE0C	ohne UART und Stack-Pointer
IN	FE43	Eingang in Adressmode
INNER	FE77	Eingang in DATA-Mode
OTHER	FE80	Eingabe ASCII-Zeichen von der Audio Cassette über UART
LEGAL	FE93	Bringt ASCII-Zahl, wenn 0–9 oder A bis F. Sonst FF
INPUT	FEED	Eingabe ASCII-Zeichen vom Keyboard.

Wir wollen wieder sofort mit einem Beispiel beginnen.

Problemstellung:

Wir wollen ein Programm erstellen, welches ein Zeichen, das über die Tastatur eingegeben wird, erkennt, aufnimmt und auf dem Bildschirm darstellt.

Flußdiagramm:



Wir müssen zuerst einmal den Akkumulator mit einem bestimmten Wert von der Tastatur laden. Da gibt es bereits eine Routine die dies für uns erledigen könnte. Sie beginnt im Speicher an der Adresse $FD00. Diese Routine lädt den ASCII-Wert des Zeichens, welches wir auf der Tastatur gedrückt haben in den Akkumulator. Was wir nun noch brauchen ist eine Routine, die den Wert auf den Bildschirm bringt. Wir könnten jetzt einen Bildschirmspeicherplatz direkt mit unserem Zeichen laden und hätten somit unser Zeichen bereits dargestellt.

Es gilt jedoch auch wieder eine andere Möglichkeit. Wir können aber auch nach einer Routine suchen, die bereits vorhanden ist und die Aufgabe erledigt, die wir lösen möchten. Wir finden eine solche Routine an der Adresse $BF2D.

Diese Routine kann sogar noch etwas mehr als wir eigentlich wollten. Sie ermöglicht uns u. a. mehr als nur ein Zeichen darzustellen, Verschieben des Zeichens sowie noch einige andere interessante Operationen.

Lösung unseres kleinen Problems:

```
0280 2000FD JSR  $FD00
0283 202DBF JSR  $BF2D
0286 4C8002 JMP  $0280
```

In Maschinensprache übersetzt ergibt sich dies:

2000FD 202DBF

Wenn wir wollen können wir dieses kleine Programm in einer unendlichen Schleife laufen lassen. Wir können jetzt JMP $0280 hinzufügen. Wobei wir annehmen, daß unser kleines Programm oben an der Adresse 0280 Hex beginnt.

JMP $0280

In Maschinencode sieht dieser Befehl wiefolgt aus:

4C8002

Geben Sie nun dieses kleine Programm mit Hilfe des Monitors ein:

Gehen Sie mit @ (Control P) in den Dateneingabemode des Monitors und geben Sie die richtigen Werte ein. Um das Programm auszuführen, geben Sie dann G 0280 ein. Um aus der unendlichen Schleife wieder herauszukommen drücken Sie einfach die BREAK-Taste. Dann können Sie wieder mit G 800 den Monitor starten.

Eines der größten Geheimnisse bei der Programmierung in Maschinensprache ist der Trick immer zuerst nach Routinen im Monitor (Betriebssystem etc.) zu suchen. Wenn Sie eine Routine finden, die Ihre Programmieraufgabe lösen kann, verwenden Sie sie. Sie sparen viel Codier- und Fehlersucharbeit. Aus diesem Grunde lohnt es sich immer die Monitore zuerst zu durchforsten und nachzusehen, wo man brauchbare Routine findet.

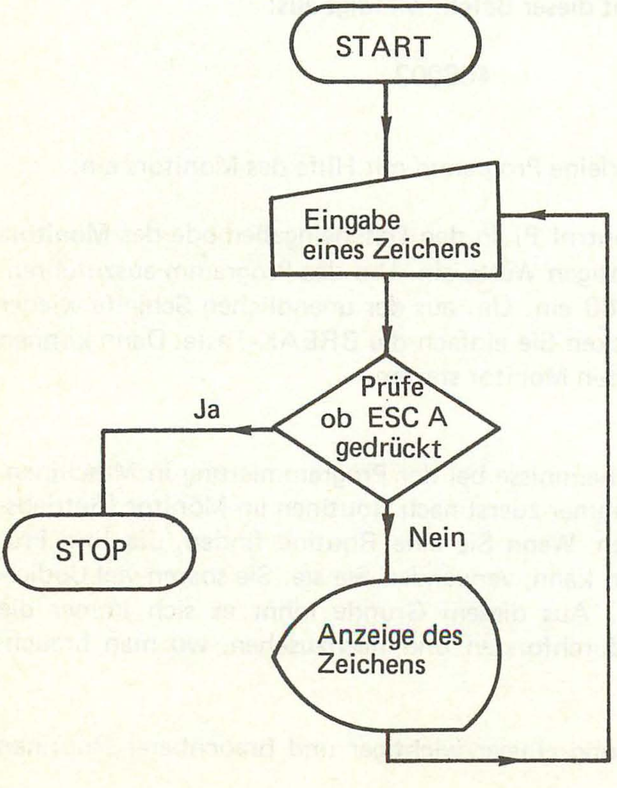
Kurze Zusammenstellung einiger wichtiger und brauchbarer Routinen

aus dem Superboard:

FD00	Eingabe eines Zeichens von der Tastatur, Ergebnis steht im Akku
FCB1	Ausgabe 1 Byte von Akku auf Cassette
FE00	Eingang in den Monitor
BF2D	Druckt Zeichen im Akku auf den Bildschirm
FE80	Bringt ASCII-Zeichen von Cassette in Akku.

Programmieren in Maschinensprache bindet Sie oft sehr an einen bestimmten Computer. Der Computer mit dem Sie arbeiten hat spezielle Routinen, wie sie nur in diesen und keinen anderen System vorkommen. Es ist ganz wichtig für Sie diese Routinen zu finden und zu lernen, sie anzuwenden.

Problem Nr. 2



Entwurf eines Flußdiagrammes

Wir haben nun unsere Routine und die endlose Schleife programmiert und sie arbeitet auch. Aber wenn wir nur kurzzeitig aus der Schleife herauswollen müssen, wir die BREAK-Taste drücken und dann unser Programm wieder neu starten. Dies ist etwas umständlich. Nun wollen wir ein Programm entwerfen, welches uns ohne BREAK-Taste aus der Schleife herauslässt.

Sie erkennen jetzt sicher gleich, daß wir ein rautenartiges Kästchen verwendet haben. Ein Pfeil geht hinein und zwei Pfeile kommen heraus. Dies ist ein "Entscheidungsblock". Dies bedeutet, daß eine Reihe von Ergebnissen im Programmablauf entstehen können und es auf diese Ergebnisse verschiedene Antworten geben kann. Meist einfach nur Ja oder Nein. Je nach Antwort des Computers reagiert dann dieser unterschiedlich. In BASIC würde dies einem Bedingungsbefehl entsprechen. Z. B.

IF ESC Then ....

IF "A" Then ....

Lösung: Startadresse 0280 Hex

```
1 JSR   $FD00
  CMP   #$41
  BEQ   $028D
2 JSR   $BF2D          (1X JSR)
  JMP   $0280
  RTS
```

Jetzt müssen wir noch eine Reihe von Dingen erläutern.

Das $-Zeichen bedeutet daß eine Hexadezimalzahl folgt. Das Nummernzeichen # sagt, daß es sich um unmittelbare Adressierung handelt.

Befehlserläuterung:

Der CMP-Befehl ist ein Vergleichsbefehl (CoMPare = Vergleichen). Wir vergleichen mit diesem Befehl den Akkumulator mit dem nächst folgenden Byte. Der Hex-Wert für das ASCII-Äquivalent von A ist 41 (Dezimal 61). Nach dem Vergleich treffen wir auf den Befehl BEQ (Branch if Equal). Diese Bedingung wird nur erfüllt, wenn ein A($41), also eine 41 Hex im Akkumulator vorhanden ist. Da die beiden dann

gleich sind, verzweigt sich der Computer auf einen anderen Speicherplatz.
Sie müssen jedoch die A-Taste drücken, um den Wert 41 Hex in den Akku zu bringen.

Sie müssen lernen, wie man einen solchen Sprung berechnet. Dazu muß man, wissen wie viele Befehle übersprungen werden müssen. In unserem Falle ist die Antwort $06. Um diese Aufgabe korrekt lösen zu können, muß auch bekannt sein wieviel Bytes jeder der Befehle benötigt.

Der JSR-Befehl benötigt 3 Byte. Es werden 2 dieser Befehle übersprungen und dann wird wieder zum CMP-Befehl zurückgekehrt und verglichen (3 Byte + 3 Byte = 06 Byte).

20	00	FD	C9	41	F0	06
20	2D	BF	4C	80	02	60

Geben Sie diesen Code ab 0280 Hex ein und Disassemblieren Sie einmal den Code zur Kontrolle.

```
0280    2000FD    JSR  $FD00
0283    C941      CMP  #$41
0285    F006      BEQ  $0280
0287    202D8F    JSR  $BF2D
028A    4C8002    JMP  $0280
028D    60        RTS
```

Maschinensprachen-Quickis

Jetzt wollen wir kleine Beispiele eingeben und etwas experimentieren.

1. Laden Sie den Monitor ein
2. Drücken Sie die BREAK-Taste
3. Geben Sie M ein
4. Geben Sie 0800G ein

5. Shift P = @ 0222 : 20 (Line Feed nach jeder Eingabe drücken !)

0223 :	00	
0224 :	FD	
0225 :	20	
0226 :	2D	
0227 :	BF	
0228 :	4C	
0229 :	22	
022A :	02	RETURN führt zurück in COMMAND MODE

6. Geben Sie Q 0222 ein. Dieser Befehl disassembliert diesen Code und zeigt folgendes:

0222	2000FD	JSR	$FD00	Eingabe
0225	202DBF	JSR	$BF20	Anzeige auf Bildschirm
0228	4C2202	JMP	$0222	Sprung an den Anfang

Diese Routine ermöglicht es Ihnen etwas über die Tastatur einzugeben und es auf dem Bildschirm darzustellen. Drücken Sie nun RETRURN und geben Sie G0222 ein. Alles was Sie nun über die Tastatur eingeben, erscheint auf dem Bildschirm. Um aus den Programm wieder herauszukommen, müssen Sie BREAK drücken. Drücken Sie nun wieder M und 0800G.

Ein etwas kompliziertes Programm

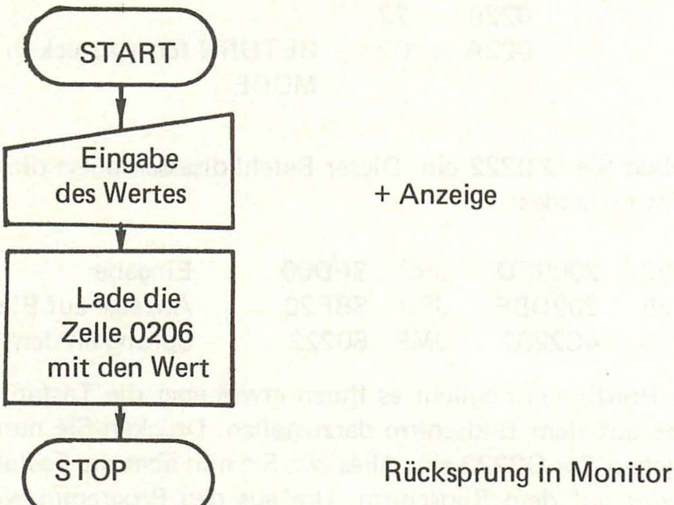
Nun wollen wir uns ein anderes Programm ansehen. Nehmen wir einmal an, wir wollen es ermöglichen, daß die Geschwindigkeit in der die Zeichen auf dem Bildschirm erscheinen veränderbar ist. Wir wollen das Programm für unsere Anwendunge benutzen. Mit dem Monitor geht dies ganz einfach. Alles, was wir tun müssen, ist folgendes:

1. CRTLP @ 0206
2. Wir geben einen Wert zwischen 00 und FF in diese Zelle (FF ist die niedrigste Geschwindigkeit).
3. Geben Sie in Zelle 0206 FF ein.

Probieren Sie es einmal aus und disassemblieren Sie zur Kontrolle einen beliebigen Bereich (z. B. ab Adresse 0800).
Wechseln Sie die Werte etwas und beobachten Sie die Veränderungen.

Nun wollen wir ein Programm schreiben, welches die Geschwindigkeit für uns ändert. Was wir hierzu brauchen ist eine Eingaberoutine die unsere Eingabe von der Tastatur aufnimmt und diese in die Speicherzelle $0206 schreibt.

Das zugehörige Flußdiagramm sieht wiefolgt aus:



Dies sieht so aus, als ob es funktionieren könnte. Wird es aber auch wirklich arbeiten?
Was geschehen wird, ist, daß die Tastaturroutine den Wert als ASCII-Wert in die Speicherzelle gibt. Wir müssen also ein Zeichen über die Tastatur eingeben, dessen Wert aus der ASCII-Tabelle dem gewünschten Wert in Zelle 0206 (Hex) entsprechen soll.

Um die Sache auf dem Bildschirm verfolgen zu können, springen wir in die Anzeigenroutine. Am Schluß kehren wir dann in den Monitor des Superboardes zurück (Adresse FE00 Hex).

So sieht das Programm aus:

```
JSR   $FD00    ;  Eingaberoutine
JSR   $BF2D    ;  Anzeigeroutine (nur ganz kurz)
STA   $0206    ;  Bringe Akku in Zelle 206
JMP   $FE00    ;  Springe zurück in Monitor
```

Das Ausrufezeichen (!) hat lt. ASCII-Tabelle den Wert 33. Wenn wir also (!) eingeben, kommt die Zahl 33 in die Speicherzelle 0206 (21 Hex = 33 dezimal).

An die richtigen Adressen gebracht ergibt:

0280 2000FD
0283 202DBF
0286 8D0602
0289 4C00FE

Starten Sie das Programm an Adresse 0280, geben Shift - Ausrufezeichen (!) ein und testen Sie ob in Zelle 0206 auch 21 steht.

Das obige Programm nimmt den Wert, der sich gerade im Akku befindet und bringt ihn in Zelle 0206. Dann springt das Programm sofort wieder in den Monitor des Superboards (nicht Supermonitor).

Für diejenigen, die nur mit dem im ROM enthaltenen Monitor arbeiten, hier noch einmal kurz die Befehle:

Schrägstrich (/) Zeichen = Wechsel in Data-Mode
Punkt (.) = Wechsel in den Adress-Mode
L = Laden von Cassette
G = Starten eines Maschinenprogrammes an der gerade im Adressfeld angezeigten Adresse.

8.5 AIM 65 (PC 100)

Der AIM 65 von Rockwell ist ein Einplatinencomputer mit Schreibmaschinentastatur, Drucker und einem alphanumerischen LED-Display mit jeweils 20 Zeichen.

8.51 Miniassembler

Nach dem Einschalten und betätigen der RESET-Taste ist der AIM 65 im Monitor. Über diesen Monitor kann in Assemblerschreibweise ein Maschinenprogramm eingegeben werden. Mit I wird in dieses Eingabeprogramm gesprungen und vom Rechner die augenblickliche Adresse angezeigt.

Mit * kann eine neue Anfangsadresse gesetzt werden.

Beispiel:
Eingabe von Assemblertext ab Adresse $900 (nur die unterstrichenen Zeichen müssen eingegeben werden)

```
I
A004    *=   0900  ⟨RET⟩
0900    JSR  E93C  ⟨RET⟩   (READ)
0903    STA  800   ⟨RET⟩   Diese Zeile wird nicht angenommen,
                          sondern:
0903    STA  0800  ⟨RET⟩
0906    JSR  E97A  ⟨RET⟩   (OUTPUT)
0909    BRK
090A    ⟨ESC⟩              Damit wird der Miniassembler ver-
                          lassen.
```

Wird bei der Eingabe in den Miniassembler ein Fehler gemacht, so wird die Zeile nicht angenommen. Die Adresse bleibt ungeändert. Alle Adressen müssen als 4-stellige Hexadezimalzahl eingegeben werden. Das $-Zeichen vor einer Adresse oder einer Zahl ist nicht erlaubt, da alle eingegebenen Daten Hexadzimal interpretiert werden. Die Eingabe wird mit der ESC-Taste beendet.

Der Zelleninhalt kann durch M ausgegeben werden

```
⟨M⟩=   0900 ⟨RET⟩   20   82   EC   8D   ⟨LT⟩ (Leertaste)
⟨ ⟩    0904  00     08   20   7A   ⟨LT⟩
⟨ ⟩    0908  E9     00   12   0B
```

Eine Rückübersetzung des Maschinencodes wird durch Eingabe von K eingeleitet.

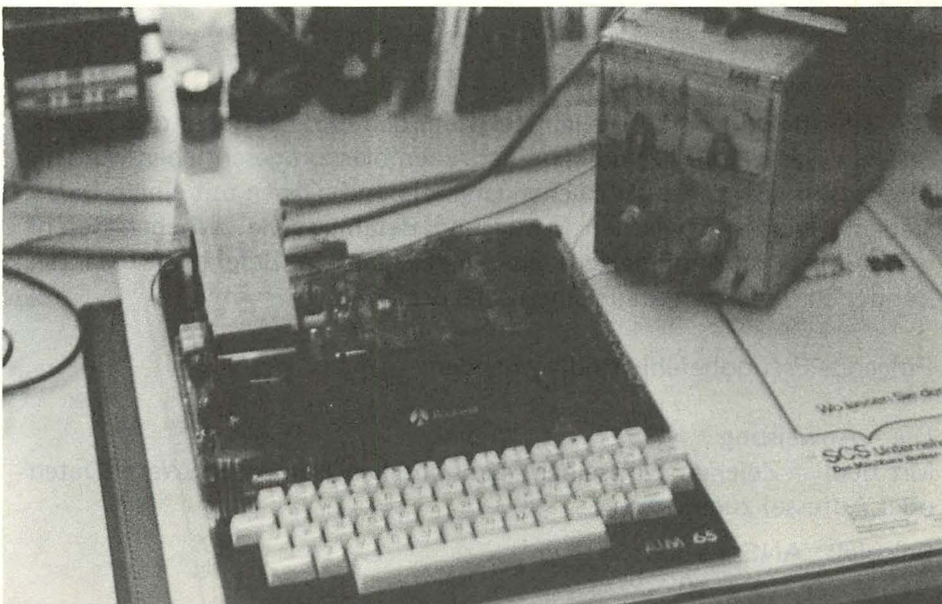
```
⟨K⟩ * = 0900 ⟨RET⟩
/
```

Jetzt muß die Zahl der zu disassemblierenden Zeilen eingegeben werden. Jede Zahl zwischen 01 und 99 ist möglich.

Beispiel:
Rückübersetzung des bei $900 eingegebenen Maschinencodes

⟨K⟩ * = 0900 ⟨RET⟩
/04

```
<K>*=0900
/04
 0900 20 JSR E93C
 0903 8D STA 0800
 0906 20 JSR E97A
 0909 00 BRK
```



Der AIM 65

8.52 AIM - Assembler

Der ROM residente Assembler des AIM 65 ist ein 2 PASS Assembler. Er übersetzt den Assemblertext, der entweder im RAM in einem Textspeicher abgelegt ist, oder der von Kassette geladen wird.

Eingabe von Assemblertext über den Texteditor: Die Eingabe von E im Monitor ruft den Texteditor auf. Danach wird die Anfangs- und Endadresse des Textspeichers festgelegt.

FROM = <u>0200</u> ⟨ <u>RET</u> ⟩ TO = <u>0400</u> ⟨ <u>RET</u> ⟩

Diese Angaben legen den Textspeicher in den Speicherbereich von $200 bis $400 und reservieren somit Platz für 512 Byte. Das entspricht etwa einem Text von 512 Buchstaben.
Danach folgt die Abfrage, ob Text von einem externen Gerät gelesen werden soll.

IN = ⟨ <u>RET</u> ⟩

Die Eingabe von ⟨ RET ⟩ allein zeigt an, daß der Text über das Tastenfeld eingegeben wird. Nun kann der Assemblertext geschrieben werden. Marken können bis zu 6 Zeichen lang sein und müssen mit einem Buchstaben beginnen. Zwischen Marke und Befehl, sowie zwischen Befehl und Adresse muß mindestens ein Leerzeichen sein. Befehle und Marken können in der ersten Stelle einer Zeile beginnen.

Folgende Pseudobefehle sind vorhanden:

Die = Anweisung
Mit dem = Zeichen wird einem symbolischen Namen ein Wert (Daten oder Adresse) zugewiesen.

```
Beispiel:  ANF = $800
             Z = 10
            BO = %00100000
```

ANF entspricht der Hexadezimalzahl 800, Z entspricht der Dezimalzahl 10 und BO ist die Binärzahl 0 0 1 0 0 0 0 0. Mit der * Anweisung wird der Programmzähler gesetzt.

* = $900

legt den Beginn des folgenden Programms auf die Adresse $900. Damit können auch Speicherbereiche für Daten reserviert werden.

```
       * = $800
ADRE = * + $10
ENDE = * + $20
```

Mit dieser Anweisungsfolge entspricht ADDE der Adresse $810 und ENDE der Adresse $820.

Die .BYTE Anweisung
Diese Anweisung belegt Speicherplatz für die angegebenen Daten.

Beispiele:

TEXT .BYTE 'ABCD', OD, FF

Damit wird an dieser Stelle unter dem Namen Text die Buchstabenfolge ABCD als ASCII-Zeichen und die beiden Hexadezimalzahlen OD und FF gespeichert. Mehrfache Daten werden durch Komma getrennt. Eine Marke ist nicht unbedingt erforderlich.

Die .WORD Anweisung
Durch die .WORD Anweisung werden 4 stellige Hexadezimalzahlen im Adressenformat gespeichert. Mit

.WORD $ABCD

wird $ABCD im Rechner als CD AB in zwei aufeinanderfolgenden Speicherplätzen abgelegt. Damit können Daten für Spruntabellen eingegeben werden. Mehrfache Datenangaben werden durch Komma getrennt.

Die .DBYTE Anweisung
Diese Anweisung entspricht der .WORD Anweisung, nur werden hier die Daten in der angegebenen Reihenfolge gespeichert.

.DBYTE $ABCD, $3A4F

wird im Rechner als AB CD 3A 4F abgelegt.

Die folgenden Pseudobefehle sind Anweisungen für das Assemblerprotokoll.

Die .PAGE Anweisung gibt eine Seitenüberschrift aus.

Beispiel: .PAGE 'PROG.1'

Die .SKIP Anweisung erzeugt in Assemblerprotokoll eine Leerzeile.

Die .END Anweisung gibt das Ende des Quelltextes an.

Die .OPT Anweisung kann 3 Parameter enthalten.

LIST, NOLIST	Ausgabe oder Unterdrückung der Ausgabe des übersetzten Textes.
GENERATE, NOGENERATE	Mit NOGENERATE werden bei einer .BYTE Anweisung nur die beiden ersten Datenbytes ausgegeben.
ERRORS, NOERRORS	Die Ausgabe von Fehlermeldungen wird entweder zugelassen oder unterbunden.

Von den Parametern werden nur die ersten 3 Buchstaben erkannt. Damit kann eine .OPT Anweisung in folgender Form angegeben werden: .OPT NOL, NOG, ERR

Wird keine .OPT Anweisung verwendet, so gilt:

.OPT LIS, NOG, ERR

Die .FILE Anweisung
Längere Programme können in Teile zerlegt und unter einem Namen auf Kassette gespeichert werden. Für die Übersetzung muß aber der vollständige Text im Rechner sein. Durch die .FILE 〈 NAME 〉 An-

weisung am Ende eines Programmteils wird das Einlesen des Textes ⟨ NAME ⟩ von Kassette eingeleitet.

Nach dem Schreiben des Programms wird mit der ESC-Taste der Texteditor verlassen und in den Monitor zurückgesprungen. Der Assembler wird mit N aufgerufen.

⟨ N . Danach erfolgt die Eingabe der Anfangs- und Endadresse für die Symbol-Tabelle

FROM = 400 ⟨ RET ⟩ TO = 500 ⟨ RET ⟩

Diese Adressen wird man zweckmässigerweise im Anschluß an den Textspeicher legen. Dabei muß man aber beachten, daß das übersetzte Programm nicht den Textspeicher oder den Speicherbereich für die Symbol-Tabelle benutzt. Als nächstes muß dem Assembler angegeben werden, wo sich der Quelltext befindet. Mit

IN = M

wird der Quelltext aus dem Textspeicher übersetzt. Wird die nächste Anfrage

LIST? mit Y beantwortet, so wird mit LIST-OUT=P ein Übersetzungsprotokoll auf den Drucker ausgegeben. Danach muß noch angegeben werden, wohin der Maschinencode geschrieben wird. Wird auf OBJ? ein N eingegeben, so wird das Programm an der, durch die * = Anweisung angegebenen Adresse gespeichert.

Sind bei der Übersetzung Fehler aufgetreten, so können diese im Textspeicher korrigiert werden. Die Rückkehr in den Texteditor geschieht durch Eingabe von T. Es wird die 1. Zeile des Programms angezeigt. Soll diese geändert werden, so wird sie mit K gelöscht und nach der Eingabe von I neu eingegeben. Mit U wird die vorangegangene Zeile angezeigt.

Abbildung 8.5 zeigt ein kleines Beispiel für ein AIM-Programm. Bei

Betätigen der Funktionstaste F1 wird ein bei MSG1 gespeicherter Text ausgegeben.

```
OBJ?N
PASS 1

PASS 2

==00A0
         *=$010C
==010C
4C0008 JMP F1

==010F OUTPUT=$E97A

==010F CLR=$EB44

==010F
         *=$0800
==0800 F1
2044EB JSR CLR
A2FF   LDX #$FF
==0805 LOOP
E8     INX
BD1408 LDA MSG1,X
C93B   CMP #';'
F006   BEQ RET
207AE9 JSR OUTPUT
18     CLC
90F2   BCC LOOP
==0813 RET
60     RTS
;
==0814 MSG1
20454C .BYTE ' EL'
434F   .BYTE 'COMP;'
       .END
  ERRORS= 0000
```

Abbildung 8.5

Die wichtigsten Monitorunterprogramme sind:

OUTPUT	Anfangsadresse $E97A Ausgabe eines ASCII-Zeichens auf LED-Display und Drucker.
NUMA	Anfangsadresse $EA46 Druckt den Inhalt des Akkumulators als 2 Hexadezimalzahlen aus.
READ	Anfangsadresse $E93C Liest ein Zeichen vom Tastenfeld in den Akkumulator.

NOTIZEN

Disassembler

9.0 Disassembler

Das Gegenstück zum Assembler, der Assemblertext in Maschinencode umwandelt, ist der Disassembler.

Dieser wandelt, soweit es möglich ist, den Maschinencode wieder in Assemblertext zurück. Diese Rückübersetzung ist dann notwendig, wenn ein Programm nur als Maschinencode vorliegt und es analysiert werden soll. Die Rückübersetzung ist aber auch dann notwendig, wenn Fehler im Programm gefunden werden sollen.

9.1 Disassemblieren von Maschinencode

Die Rückübersetzung von Maschinencode ist nicht ganz problemlos Sie muß, um überhaupt erfolgreich zu sein, auf dem Speicherplatz eines gültigen Operationscodes beginnen. Das Programm entschlüsselt diesen Operationscode und stellt fest, ob es ein 1, 2 oder 3 Byte Befehl ist. Dann wird die Assemblerschreibweise des Befehls aus einer Tabelle geholt, die Art der Adressierung festgestellt und die Befehlszeile ausgegeben. Wenn dies ein 3 Byte Befehl war, so muß das 3. Byte nach dem gerade entschlüsselten Befehl wieder ein Operationscode sein. Sind nun Daten im Programm vorhanden, so versucht der Disassembler, diese ebenfalls zu entschlüsseln. Dabei kann der Disassembler zu falschen Rückübersetzungen gelangen. In Abbildung 9.1 ist dies gezeigt.

Das Programm ist in Speicherzelle $C44E mit dem Sprung nach $C100 (4C 00 01) beendet. Es folgen Daten. Diese werden teilweise als Befehlszeilen interpretiert, wie z. B. AD F9 99 als LDA $99F9, oder 2C BA E8 als BIT $E8BA.

Dabei geht aber bei der Disassemblierung der Anfang eines anderen Programmteils in Adresse $C460 verloren. Das letzte Byte des rückübersetzten Befehls BIT $E8BA ist E8, und dies ist ein gültiger Operationscode für INX.

```
C447-   68          PLA
C448-   A8          TAY
C449-   68          PLA
C44A-   AA          TAX
C44B-   4C 0C FD    JMP   $FD0C
C44E-   4C 00 C1    JMP   $C100
C451-   C0 EA       CPY   #$EA
C453-   EA          NOP
C454-   EA          NOP
C455-   AD F9 99    LDA   $99F9
C458-   A7          ???
C459-   4C 49 65    JMP   $6549
C45C-   9E          ???
C45D-   47          ???
C45E-   2C BA E8    BIT   $E8BA
C461-   C8          INY
C462-   98          TYA
C463-   E8          INX
C464-   98          TYA
C465-   18          CLC
```

9.1 Fehlerhafte Rückübersetzung

Adressen und Sprungziele werden als Hexadezimalzahlen ausgegeben. Nur sehr komfortable Disassembler können hier symbolische Namen in einem 2. Durchgang einführen.

In Abbildung 9.2 ist der im APPLE II verwendete Disassembler gezeigt. Dieser wurde von Steve Wozniak geschrieben und im DDJ September 1976 veröffentlicht. Die Anfangsadresse des rückzuübersetzenden Programms wird nach $44 und $45 geschrieben und der Disassembler bei Adresse $800 gestartet.

Als Besonderheit sei darauf hingewiesen, daß der Text für einen Assemblerbefehl, der normalerweise 3 Byte belegen würde, in 2 Byte zusammengefasst wurde, um Speicherplatz zu sparen. Wenn nur die 32 Buchstaben ausgegeben werden sollen, so genügen für die Darstellung eines Buchstaben 5-bit. In 16-bit, 2 Byte, können also 3 Buchstaben untergebracht werden. In den Abbildungen 9.3 und 9.4 ist der Hexdump für das Programm und für die Tabellen getrennt angegeben.

```
0800              ;
0800              ;
0800              ;*****************************
0800              ;*                           *
0800              ;*    DISASSEMBLER           *
0800              ;*                           *
0800              ;*****************************
0800              ;
0800              ;
0800              M      EPZ $40
0800              FORMAT EPZ M
0800              LENGTH EPZ M+1
0800              LMNEM  EPZ M+2
0800              RMNEM  EPZ M+3
0800              PCL    EPZ M+4
0800              PCH    EPZ M+5
0800              COUNT  EPZ M+6
0800              ;
0800              PRBYTE EQU $FDDA
0800              CHAROU EQU $FDED
0800                     ORG $800
0800 A913         DSMBL  LDA #$13
0802 8546                STA COUNT
0804 201208       DSMBL2 JSR INSTDSP
0807 20EF08              JSR PCADJ
080A 8544                STA PCL
080C 8445                STY PCH
080E C646                DEC COUNT
0810 D0F2                BNE DSMBL2
0812 20D308       INSTDS JSR PRPC
0815 A144                LDA (PCL,X)
0817 A8                  TAY
0818 4A                  LSR
0819 900B                BCC IEVEN
081B 4A                  LSR
081C B017                BCS ERR
081E C922                CMP #$22
0820 F013                BEQ ERR
0822 2907                AND #$7
0824 0980                ORA #$80
0826 4A           IEVEN  LSR
0827 AA                  TAX
0828 BDFE08              LDA MODE,X
082B B004                BCS RTMODE
082D 4A                  LSR
082E 4A                  LSR
082F 4A                  LSR
0830 4A                  LSR
0831 290F         RTMODE AND #$F
0833 D004                BNE GETFMT
0835 A080         ERR    LDY #$80
0837 A900                LDA #$0
0839 AA           GETFMT TAX
083A BD4209              LDA MODE2,X
083D 8540                STA FORMAT
083F 2903                AND #$3
0841 8541                STA LENGTH
0843 98                  TYA
0844 298F                AND #$8F
0846 AA                  TAX
0847 98                  TYA
0848 A003                LDY #$3
084A E08A                CPX #$8A
084C F00B                BEQ MNNDX3
084E 4A           MNNDX1 LSR
084F 9008                BCC MNNDX3
0851 4A                  LSR
0852 4A           MNNDX2 LSR
0853 0920                ORA #$20
```

```
0855 88              DEY
0856 D0FA            BNE MNNDX2
0858 C8              INY
0859 88      MNNDX3  DEY
085A D0F2            BNE MNNDX1
085C 48              PHA
085D B144    PROP    LDA (PCL),Y
085F 20DAFD          JSR PRBYTE
0862 A201            LDX #$1
0864 20E608  PROPBL  JSR PRBL2
0867 C441            CPY LENGTH
0869 C8              INY
086A 90F1            BCC PROP
086C A203            LDX #$3
086E C004            CPY #$4
0870 90F2            BCC PROPBL
0872 68              PLA
0873 A8              TAY
0874 B95C09          LDA MNEML,Y
0877 8542            STA LMNEM
0879 B99C09          LDA MNEMR,Y
087C 8543            STA RMNEM
087E A900    PRMN1   LDA #$0
0880 A005            LDY #$5
0882 0643    PRMN2   ASL RMNEM
0884 2642            ROL LMNEM
0886 2A              ROL
0887 88              DEY
0888 D0F8            BNE PRMN2
088A 69BF            ADC #$BF
088C 20EDFD          JSR CHAROUT
088F CA              DEX
0890 D0EC            BNE PRMN1
0892 20E408          JSR PRBLNK
0895 A206            LDX #$6
0897 E003    PRADR1  CPX #$3
0899 D012            BNE PRADR3
089B A441            LDY LENGTH
089D F00E            BEQ PRADR3
089F A540    PRADR2  LDA FORMAT
08A1 C9E8            CMP #$E8
08A3 B144            LDA (PCL),Y
08A5 B01C            BCS RELADR
08A7 20DAFD          JSR PRBYTE
08AA 88              DEY
08AB D0F2            BNE PRADR2
08AD 0640    PRADR3  ASL FORMAT
08AF 900E            BCC PRADR4
08B1 BD4F09          LDA CHAR1-1,X
08B4 20EDFD          JSR CHAROUT
08B7 BD5509          LDA CHAR2-1,X
08BA F003            BEQ PRADR4
08BC 20EDFD          JSR CHAROUT
08BF CA      PRADR4  DEX
08C0 D0D5            BNE PRADR1
08C2 60              RTS
08C3 20F208  RELADR  JSR PCADJ3
08C6 AA              TAX
08C7 E8              INX
08C8 D001            BNE PRNTYX
08CA C8              INY
08CB 98      PRNTYX  TYA
08CC 20DAFD  PRNTAX  JSR PRBYTE
08CF 8A      PRNTX   TXA
08D0 4CDAFD          JMP PRBYTE
08D3 A98D    PRPC    LDA #$8D
08D5 20EDFD          JSR CHAROUT
08D8 A545            LDA PCH
08DA A644            LDX PCL
```

```
08DC 20CC08   140          JSR PRNTAX
08DF A9AD     141          LDA #$AD
08E1 20EDFD   142          JSR CHAROUT
08E4 A203     143   PRBLNK LDX #$3
08E6 A9A0     144   PRBL2  LDA #$A0
08E8 20EDFD   145   PRBL3  JSR CHAROUT
08EB CA       146          DEX
08EC D0F8     147          BNE PRBL2
08EE 60       148          RTS
08EF A541     149   PCADJ  LDA LENGTH
08F1 38       150   PCADJ2 SEC
08F2 A445     151   PCADJ3 LDY PCH
08F4 AA       152          TAX
08F5 1001     153          BPL PCADJ4
08F7 88       154          DEY
08F8 6544     155   PCADJ4 ADC PCL
08FA 9001     156          BCC RTS1
08FC C8       157          INY
08FD 60       158   RTS1   RTS
08FE 400245   159   MODE   HEX 40024503D00840093022
0901 03D008
0904 400930
0907 22
0908 4533D0   160          HEX 4533D008400940024533
090B 084009
090E 400245
0911 33
0912 D00840   161          HEX D0084009400245B3D008
0915 094002
0918 45B3D0
091B 08
091C 400900   162          HEX 400900224433D08C4400
091F 224433
0922 D08C44
0925 00
0926 112244   163          HEX 11224433D08C449A1022
0929 33D08C
092C 449A10
092F 22
0930 4433D0   164          HEX 4433D008400910224433
0933 084009
0936 102244
0939 33
093A D00840   165          HEX D008400962137 8A9
093D 096213
0940 78A9
0942 002181   166   MODE2  HEX 002181820000594D9192
0945 820000
0948 594D91
094B 92
094C 864A85   167          HEX 864A859D
094F 9D
0950 ACA9AC   168   CHAR1  HEX ACA9ACA3A8A4
0953 A3A8A4
0956 D900D8   169   CHAR2  HEX D900D8A4A400
0959 A4A400
095C 1C8A1C   170   MNEML  HEX 1C8A1C235D8B1BA19D8A
095F 235D8B
0962 1BA19D
0965 8A
0966 1D239D   171          HEX 1D239D8B1DA1002919AE
0969 8B1DA1
096C 002919
096F AE
0970 69A819   172          HEX 69A8192324531B232453
0973 232453
0976 1B2324
0979 53
097A 19A100   173          HEX 19A1001A5B5BA5692424
```

```
097D 1A5B5B
0980 A56924
0983 24
0984 AEAEA8  174        HEX AEAEA8AD29007C00159C
0987 AD2900
098A 7C0015
098D 9C
098E 6D9CA5  175        HEX 6D9CA569295384133411
0991 692953
0994 841334
0997 11
0998 A56923  176        HEX A56923A0
099B A0
099C D8625A  177  MNEMR HEX D8625A48266294885444
099F 482662
09A2 948854
09A5 44
09A6 C85468  178        HEX C8546844E89400B40884
09A9 44E894
09AC 00B408
09AF 84
09B0 74B428  179        HEX 74B4286E74F4CC4A72F2
09B3 6E74F4
09B6 CC4A72
09B9 F2
09BA A48A00  180        HEX A48A00AAA2A274747472
09BD AAA2A2
09C0 747474
09C3 72
09C4 4468B2  181        HEX 4468B232B20022001A1A
09C7 32B200
09CA 22001A
09CD 1A
09CE 262672  182        HEX 262672728 8C8C4CA2648
09D1 7288C8
09D4 C4CA26
09D7 48
09D8 4444A2  183        HEX 4444A2C8
09DB C8
             184        END
```

9.2 Programm Disassembler

```
0800- A9 13 85 46 20 12 08 20
0808- EF 08 85 44 84 45 C6 46
0810- D0 F2 20 D3 08 A1 44 A8
0818- 4A 90 0B 4A B0 17 C9 22
0820- F0 13 29 07 09 80 4A AA
0828- BD FE 08 B0 04 4A 4A 4A
0830- 4A 29 0F D0 04 A0 80 A9
0838- 00 AA BD 42 09 85 40 29
0840- 03 85 41 98 29 8F AA 98
0848- A0 03 E0 8A F0 0B 4A 90
0850- 08 4A 4A 09 20 88 D0 FA
0858- C8 88 D0 F2 48 B1 44 20
0860- DA FD A2 01 20 E6 08 C4
0868- 41 C8 90 F1 A2 03 C0 04
0870- 90 F2 68 A8 B9 5C 09 85
0878- 42 B9 9C 09 85 43 A9 00
0880- A0 05 06 43 26 42 2A 88
0888- D0 F8 69 BF 20 ED FD CA
0890- D0 EC 20 E4 08 A2 06 E0
0898- 03 D0 12 A4 41 F0 0E A5
08A0- 40 C9 E8 B1 44 B0 1C 20
08A8- DA FD 88 D0 F2 06 40 90
08B0- 0E BD 4F 09 20 ED FD BD
08B8- 55 09 F0 03 20 ED FD CA
```

```
08C0- D0 D5 60 20 F2 08 AA E8
08C8- D0 01 C8 98 20 DA FD 8A
08D0- 4C DA FD A9 8D 20 ED FD
08D8- A5 45 A6 44 20 CC 08 A9
08E0- AD 20 ED FD A2 03 A9 A0
08E8- 20 ED FD CA D0 F8 60 A5
08F0- 41 38 A4 45 AA 10 01 88
08F8- 65 44 90 01 C8 60
*
```

9.3 Hexdump Disassembler

```
08FE- 40 02
0900- 45 03 D0 08 40 09 30 22
0908- 45 33 D0 08 40 09 40 02
0910- 45 33 D0 08 40 09 40 02
0918- 45 B3 D0 08 40 09 00 22
0920- 44 33 D0 8C 44 00 11 22
0928- 44 33 D0 8C 44 9A 10 22
0930- 44 33 D0 08 40 09 10 22
0938- 44 33 D0 08 40 09 62 13
0940- 78 A9 00 21 81 82 00 00
0948- 59 4D 91 92 86 4A 85 9D
0950- AC A9 AC A3 A8 A4 D9 00
0958- D8 A4 A4 00 1C 8A 1C 23
0960- 5D 8B 1B A1 9D 8A 1D 23
0968- 9D 8B 1D A1 00 29 19 AE
0970- 69 A8 19 23 24 53 1B 23
0978- 24 53 19 A1 00 1A 5B 5B
0980- A5 69 24 24 AE AE A8 AD
0988- 29 00 7C 00 15 9C 6D 9C
0990- A5 69 29 53 84 13 34 11
0998- A5 69 23 A0 D8 62 5A 48
09A0- 26 62 94 88 54 44 C8 54
09A8- 68 44 E8 94 00 B4 08 84
09B0- 74 B4 28 6E 74 F4 CC 4A
09B8- 72 F2 A4 8A 00 AA A2 A2
09C0- 74 74 74 72 44 68 B2 32
09C8- B2 00 22 00 1A 1A 26 26
09D0- 72 72 88 C8 C4 CA 26 48
09D8- 44 44 A2 C8
*
```

9.4 Hexdump Tabellen

9.2 Suchen von ASCII-Zeichen

Das Programm in Abbildung 9.5 dient zum Auffinden von Text in Maschinencode. Damit können Daten in unbekannten Maschinenprogrammen gefunden werden. Die Anfangsadresse und die Endadresse des zu untersuchenden Blockes wird in die Zellen \$10, \$11 bzw. \$12 und \$13 geschrieben. Ist Bit 7 von MAPFLG = 1, so wird ein Hexdump des Bereiches und alle Hexadezimalzahlen größer \$21 als ASCII-Zeichen ausgegeben. Mit Bit 7 von MPFLG = 0 wird nur ein Hexdump ausgegeben.

```
0800           ;
0800           ;
0800           ;*****************************
0800           ;*                           *
0800           ;*    TEXTFINDER             *
0800           ;*                           *
0800           ;*****************************
0800           ;
0800           BLKSTR EPZ $10
0800           BLKEND EPZ $12
0800           MAPFLG EPZ $14
0800           POINTR EPZ $15
0800           POINTS EPZ $17
0800           BYTCNT EPZ $19
0800           TMPFLG EPZ $1A
0800           SPACNT EPZ $1B
0800           ;
0800           PRTCHR EQU $FDED
0800           PRTBYT EQU $FDDA
0800           MONITO EQU $FF59
0800           ;
0800           ;
0800 A510      MEMDMP LDA BLKSTR
0802 8515             STA POINTR
0804 A511             LDA BLKSTR+1
0806 8516             STA POINTR+1
0808 A9FF      ONELIN LDA #$FF
080A 851A             STA TMPFLG
080C A98D             LDA #$8D
080E 20EDFD           JSR PRTCHR
0811 A516             LDA POINTR+1
0813 8518             STA POINTS+1
0815 20DAFD           JSR PRTBYT
0818 A515             LDA POINTR
081A 8517             STA POINTS
081C 20DAFD           JSR PRTBYT
081F A900             LDA #0
0821 8519             STA BYTCNT
0823 A903      SEGMNT LDA #3
0825 851B             STA SPACNT
0827 A9A0      SPACES LDA #$A0
0829 20EDFD           JSR PRTCHR
082C C61B             DEC SPACNT
082E D0F7             BNE SPACES
0830 C619             DEC BYTCNT
0832 10EF             BPL SEGMNT
0834 A908             LDA #8
0836 8519             STA BYTCNT
0838 A000      ONEBYT LDY #0
083A B115             LDA (POINTR),Y
083C 241A             BIT TMPFLG
083E 300C             BMI PRTHEX
0840 C921             CMP #$21
0842 3008             BMI PRTHEX
0844 20EDFD           JSR PRTCHR
0847 18               CLC
0848 9005             BCC INCPNT
084A 10BC      NXTLIN BPL ONELIN
084C 20DAFD    PRTHEX JSR PRTBYT
084F E615      INCPNT INC POINTR
0851 D003             BNE ENDTST
0853 E616             INC POINTR+1
0855 D8               CLD
0856 A515      ENDTST LDA POINTR
0858 C512             CMP BLKEND
085A A516             LDA POINTR+1
085C E513             SBC BLKEND+1
085E 9017             BCC SEGTST
0860 2414             BIT MAPFLG
```

```
0862 1021                 BPL FINISH
0864 241A                 BIT TMPFLG
0866 101D                 BPL FINISH
0868 C619                 DEC BYTCNT
086A E61A         DOCMAP  INC TMPFLG
086C A517                 LDA POINTS
086E 8515                 STA POINTR
0870 A518                 LDA POINTS+1
0872 8516                 STA POINTR+1
0874 18                   CLC
0875 90AC                 BCC SEGMNT
0877 C619         SEGTST  DEC BYTCNT
0879 D0BD                 BNE ONEBYT
087B 2414                 BIT MAPFLG
087D 10CB                 BPL NXTLIN
087F 241A                 BIT TMPFLG
0881 30E7                 BMI DOCMAP
0883 10C5                 BPL NXTLIN
0885 A98D         FINISH  LDA #$8D
0887 20EDFD               JSR PRTCHR
088A 2059FF               JSR MONITO
                          END
```

9.5 Programm Textfinder

```
0800- A5 10 85 15 A5 11 85 16
0808- A9 FF 85 1A A9 8D 20 ED
0810- FD A5 16 85 18 20 DA FD
0818- A5 15 85 17 20 DA FD A9
0820- 00 85 19 A9 03 85 1B A9
0828- A0 20 ED FD C6 1B D0 F7
0830- C6 19 10 EF A9 08 85 19
0838- A0 00 B1 15 24 1A 30 0C
0840- C9 21 30 08 20 ED FD 18
0848- 90 05 10 BC 20 DA FD E6
0850- 15 D0 03 E6 16 D8 A5 15
0858- C5 12 A5 16 E5 13 90 17
0860- 24 14 10 21 24 1A 10 1D
0868- C6 19 E6 1A A5 17 85 15
0870- A5 18 85 16 18 90 AC C6
0878- 19 D0 BD 24 14 10 CB 24
0880- 1A 30 E7 10 C5 A9 8D 20
0888- ED FD 20 59 FF ED
*
```

9.6 Hexdump Textfinder

9.3 Umsetzen von Maschinenprogrammen auf andere 6502 Systeme

Das Umsetzen eines Maschinenprogramms von einem 6502 System auf ein anderes 6502 bereitet im allgemeinen keine großen Schwierigkeiten. Kann in beiden Systemen der gleiche Speicherbereich benutzt werden, so brauchen nicht einmal die Adressen umgerechnet werden. Soll das Programm in einen anderen Speicherbereich geschrieben werden, so empfiehlt es sich, das Programm mit einem RELOCATOR umzusetzen. Man spart dadurch Arbeit.

Als nächstes müssen Monitorunterprogramme gesucht werden. Diese sind für jeden Rechner verschieden. Die Bildschirmausgabe oder die Tasteneingabe erfolgt bei jedem Rechner durch ein anderes Unterprogramm. Einige Adressen sind in Kapitel 8 angegeben worden. Allerdings ist es manchmal notwendig, Teile des Monitorprogrammes vom alten Rechner auf den neuen Rechner zu übernehmen.

Beim Programmieren kann man allerdings die Arbeit des Umsetzen auf ein anderes System dadurch erleichtern, daß man Monitor-Unterprogramme nur an wenigen Stellen, oft nur an einer einzigen Stelle im Programm anspringt.

Sind z. B. in einem Programm an mehreren Stellen Ausgaben auf den Bildschirm notwendig, so wird man nicht an jeder Stelle einen Monitoraufruf JSR OUTCHR machen, sondern einen Unterprogrammsprung JSR OUT wie in folgenden Beispiel:

Nicht:	Sondern:
.	.
.	.
JSR OUTCHR	JSR OUT
.	.
.	.
.	.
JSR OUTCHR	JSR OUT
.	.
.	.
.	.
.	.
JSR OUTCHR	JSR OUT
.	.
.	.
.	.
	OUT JMP OUTCHR
	.
	.
	.

Im rechten Programm braucht bei einer Umsetzung auf ein anderes System nur eine einzige Adresse geändert werden.

Eine weitere Schwierigkeit beim Umsetzen besteht darin, daß die Monitore unterschiedliche Zellen in der Zero-Page benutzen. Falls

vom Programm die gleichen Zellen benutzt werden, so kann dies zu Fehlern führen. Falls man keinen Belegungsplan für die Zero-Page besitzt, kann man sich folgendermaßen helfen. Man beschreibt die Zero-Page mit einem beliebigen Bitmuster, z. B. $AA, nicht $00 oder $FF und ruft das Unterprogramm auf. Danach kontrolliert man durch einen Speicherauszug welche Zellen sich geändert haben. Diesen Test muß man unter Umständen mit einen anderen Bitmuster wiederholen.

Als Beispiel für das Umsetzen eines Programmes von einem Rechner auf einen anderen Rechner soll ein Programm für den KIM auf den APPLE II übertragen werden.

Das KIM-Programm berechnet den Wochentag von Beginn des gregorianischen Kalenders am 14.09.1752 bis zum 31.12.1999.

Das Programm liegt als Speicherauszug vor und belegte dort die Zeropage. Da diese im APPLE freigehalten werden soll, wird der Hexdump, bei der Adresse 800 beginnend in den Rechner eingetippt (Abbildung 9.7). Die ersten 5 Zellen, die im Original in der Zero-Page liegen, sind Hilfszellen. Das Programm beginnt bei Adresse $805.

```
0800- 00 00 00 00 00 F8 38 A9
0808- 00 85 00 85 A0 85 B3 AA
0810- A5 04 C9 00 F0 18 C9 04
0818- 90 14 A8 8A 18 69 01 AA
0820- 98 38 E9 04 D0 F0 A5 01
0828- C9 03 B0 02 C6 B3 A5 03
0830- C9 20 B0 67 C9 19 F0 2C
0838- C9 18 90 06 A9 02 85 00
0840- D0 22 C9 17 90 55 A5 04
0848- C9 53 90 06 A9 04 85 00
0850- D0 12 C9 52 D0 45 A5 01
0858- C9 09 90 3F A5 02 C9 14
0860- 90 39 B0 E8 8A 18 65 00
0868- 65 02 A6 01 75 A0 24 B3
0870- F0 03 38 E9 01 18 65 04
0878- 90 02 C6 A0 38 B0 02 E9
0880- 07 C9 07 B0 FA 24 A0 10
0888- 07 E6 A0 18 69 01 D0 EF
0890- 85 00 A5 00 85 FA 85 FB
0898- 4C 4F 1C A9 88 D0 F1 00
08A0- 00 01 04 04 00 02 05 00
08A8- 03 06 20 70 09 A2 00 BD
08B0- 10 09 20 FD
*
```

9.7 Eingegebenes KIM-Programm

Für eine weitere Untersuchung wird das Programm dissassembliert (Abbildung 9.8).

```
0800-   00      BRK
0801-   00      BRK
0802-   00      BRK
0803-   00      BRK
0804-   00      BRK
0805-   F8      SED
0806-   38      SEC
0807-   A9 00   LDA   #$00
0809-   85 00   STA   $00
080B-   85 A0   STA   $A0
080D-   85 B3   STA   $B3
080F-   AA      TAX
0810-   A5 04   LDA   $04
0812-   C9 00   CMP   #$00
0814-   F0 18   BEQ   $082E
0816-   C9 04   CMP   #$04
0818-   90 14   BCC   $082E
081A-   A8      TAY
081B-   8A      TXA
081C-   18      CLC
081D-   69 01   ADC   #$01
081F-   AA      TAX
0820-   98      TYA
0821-   38      SEC
0822-   E9 04   SBC   #$04
0824-   D0 F0   BNE   $0816
0826-   A5 01   LDA   $01
0828-   C9 03   CMP   #$03
082A-   B0 02   BCS   $082E
082C-   C6 B3   DEC   $B3
082E-   A5 03   LDA   $03
0830-   C9 20   CMP   #$20
0832-   B0 67   BCS   $089B
0834-   C9 19   CMP   #$19
0836-   F0 2C   BEQ   $0864
0838-   C9 18   CMP   #$18
083A-   90 06   BCC   $0842
083C-   A9 02   LDA   #$02
083E-   85 00   STA   $00
0840-   D0 22   BNE   $0864
0842-   C9 17   CMP   #$17
0844-   90 55   BCC   $089B
0846-   A5 04   LDA   $04
0848-   C9 53   CMP   #$53
084A-   90 06   BCC   $0852
084C-   A9 04   LDA   #$04
084E-   85 00   STA   $00
0850-   D0 12   BNE   $0864
0852-   C9 52   CMP   #$52
0854-   D0 45   BNE   $089B
0856-   A5 01   LDA   $01
0858-   C9 09   CMP   #$09
085A-   90 3F   BCC   $089B
085C-   A5 02   LDA   $02
085E-   C9 14   CMP   #$14
0860-   90 39   BCC   $089B
0862-   B0 E8   BCS   $084C
0864-   8A      TXA
0865-   18      CLC
0866-   65 00   ADC   $00
0868-   65 02   ADC   $02
086A-   A6 01   LDX   $01
086C-   75 A0   ADC   $A0,X
086E-   24 B3   BIT   $B3
0870-   F0 03   BEQ   $0875
0872-   38      SEC
0873-   E9 01   SBC   #$01
0875-   18      CLC
0876-   65 04   ADC   $04
```

```
0878-   90 02      BCC   $087C
087A-   C6 A0      DEC   $A0
087C-   38         SEC
087D-   B0 02      BCS   $0881
087F-   E9 07      SBC   #$07
0881-   C9 07      CMP   #$07
0883-   B0 FA      BCS   $087F
0885-   24 A0      BIT   $A0
0887-   10 07      BPL   $0890
0889-   E6 A0      INC   $A0
088B-   18         CLC
088C-   69 01      ADC   #$01
088E-   D0 EF      BNE   $087F
0890-   85 00      STA   $00
0892-   A5 00      LDA   $00
0894-   85 FA      STA   $FA
0896-   85 FB      STA   $FB
0898-   4C 4F 1C   JMP   $1C4F
089B-   A9 88      LDA   #$88
089D-   D0 F1      BNE   $0890
089F-   00         BRK
08A0-   00         BRK
08A1-   01 04      ORA   ($04,X)
08A3-   04         ???
08A4-   00         BRK
08A5-   02         ???
08A6-   05 00      ORA   $00
08A8-   03         ???
08A9-   06 20      ASL   $20
08AB-   70 09      BVS   $08B6
08AD-   A2 00      LDX   #$00
08AF-   BD 10 09   LDA   $0910,X
08B2-
```

9.8 KIM-Programm disassembliert

Dabei stellt man fest, daß weitere Hilfszellen (A0 bis B3) in der Zero-Page belegt werden.

Auf der Suche nach einem Monitorunterprogramm findet man in Adresse $898 ein JSR $1C4F. Betrachtet man das Programmstück zwischen Adresse $890 und $89E, so stellt man fest, daß dies eine unendliche Schleife darstellt. Beim KIM wird dabei der Inhalt der Zelle $00 nach $FA, $FB übertragen und auf 2 7-Segmentanzeigen des Displays angezeigt. Dieses Programmstück wird durch einen JSR PRTBYT ($FDDA) gefolgt von einen BRK-Befehl ersetzt.

Da sich im Programm keine Sprungbefehle oder Unterprogrammaufrufe befinden, brauchen auch keine Adressen geändert werden, die durch die Umsetzung in einen anderen Speicherbereich notwendig gewesen wären. Hier zeigt sich der Vorteil, wenn man im Programm einen JMP ⟨ MARKE ⟩ vermeidet und man stattdessen CLC, BCC

⟨ MARKE ⟩ verwendet. Dies geht nur bei kurzen Programmen, aber überall wo es möglich ist, sollte man den Sprungbefehl durch eine relative Verzweigung ersetzen.

Das Datum wird in diesem Kalenderprogramm in die Zellen $01 bis $04 eingegeben und zwar in der amerikanischen Schreibweise. Als Beispiel nehmen wir den 27.05.1981. Die Zellen werden dann folgendermaßen belegt:

$01 = 05
$02 = 27
$03 = 19
$04 = 81

Das Ergebnis erscheint in Zelle $00 und zwar mit 01 = Sonntag, 02 = Montag bis 00 = Samstag. Abbildung 9.9 zeigt die Änderung für den APPLE-Computer.

```
85 00        STA   $00
A5 00        LDA   $00
20 DA FD     JSR   $FDDA
00           BRK
```

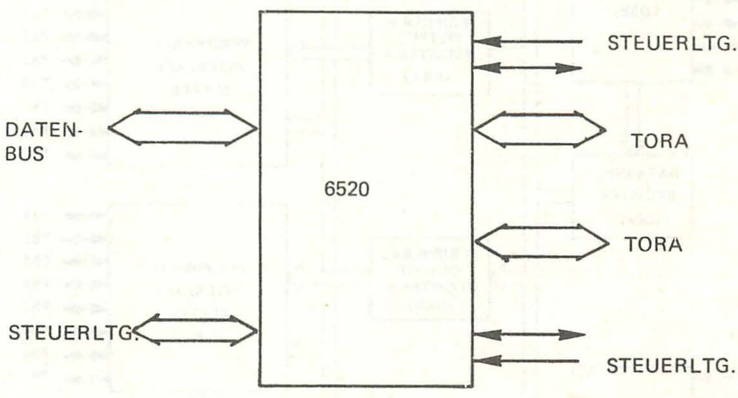
9.9 Änderung für APPLE II

Interface-Bausteine

10. Interface–Bausteine

10.1 Der Baustein PIA 6520

Der Baustein PIA (Peripheral Interface Adapter) 6520 ist ein Vorgänger des 6522, und baugleich mit dem 6820.

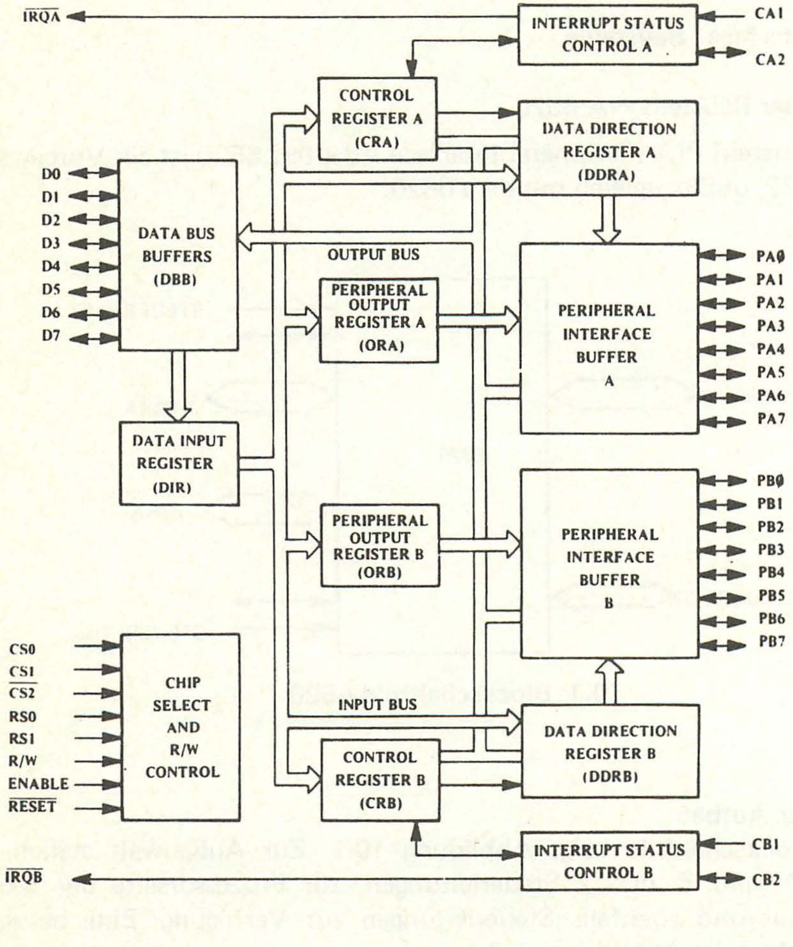


10.1 Blockschaltbild 6520

Interner Aufbau:
Das Blockschaltbild zeigt Abbildung 10.1. Zur Außenwelt stehen 2 Tore A und B und 2 Steuerleitungen, zur Prozessorseite der 8-bit Datenbus und ebenfalls Steuerleitungen zur Verfügung. Eine bessere Übersicht zeigt Abbildung 10.2.

Die beiden Tore A und B werden jeweils über die Datenrichtungsregister DDRA und DDRB und die beiden Kontrollregister CRA und CRB an-

gesteuert. Für diese 6 Register sind aber nur 2 Adressleitungen RS0 und RS1 (RS = Register Select) vorhanden. Die Datenrichtungsregister und die Tore haben die gleiche Adresse. Ein Bit im Kontrollregister (CRA2 bzw. CRB2) legt fest, welches der beiden Register angewählt wird (Abbildung 10.3).



10.2 Interner Aufbau des 6520

RS0	RS1	CRA2	CRB2	ausgewähltes Register
0	0	1	–	TORA
0	0	0	–	DDRA
0	1	–	–	CRA
1	0	–	1	TORB
1	0	–	0	DDRB
1	1	–	–	CRB

10.3 Anwahl der Register

Um alle Leitungen des Tores A z. B. als Ausgänge zu verwenden wird folgendes Programm benötigt:

```
LDA #$00     ;  Datenrichtungsregister
STA  $C0C1   ;  DDRA anwählen
LDA #$FF     ;  alle Leitungen = Eins
STA  $C0C0   ;
LDA #$04     ;  TORA anwählen
STA  $C0C1   ;
LDA #$AA     ;  Bitmuster $AA wird an
STA  $C0C0   ;  TORA ausgegeben.
```

Dabei wird angenommen, daß die Anfangsadresse des 6520 bei der Adresse $C0C0 liegt. Das Tor B wird auf die gleiche Weise über das Register CRB programmiert.

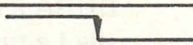
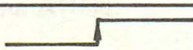
Die weitere Belegung der Bit in den Kontrollregistern zeigt Abbildung 10.4.

7	6	5	4	3	2	1	0	
IRQ A1	IRQ A2	CA2			DDRA	CA1		CRA
IRQ B1	IRQ B2	CB2			DDRB	CB1		CRB

10.4 Bitbelegung der Kontrollregister

Bit 0 und 1 steuern die Eingänge CA1 bzw. CB1. Bit 2 ist das Auswahlbit für die Datenrichtungsregister. Bit 3, 4, 5 steuern die bidirektionalen Leitungen CA2 bzw. CB2. Bit 6 und 7 sind Interrupt-Merker. Sie sind normalerweise Eins. Werden Sie durch eine externe Bedingung auf Null gesetzt, so verursacht dies ein Nullsetzen der IRQA bzw. der IRQB Leitung und somit die Auslösung eines Interrupts. Die beiden Leitungen sind offene Kollektor-Leitungen und können somit durch "wired or" verbunden werden.

CRA (CRB)

Bit 1	Bit 8	aktive Flanke an CA1 (CB1	$\overline{IRQA}$, ($\overline{IRQB}$)
0	0		Kein Interrupt
0	1		Bit 7 = 0 Interrupt
1	0		Kein Interrupt
1	1		Bit 7 = 0 Interrupt

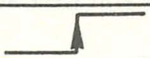
10.5 Interruptsteuerung CA1 (CB1)

In Abbildung 10.5 wird gezeigt, daß z. B. ein Interrupt durch eine negative Flanke an CA1 ausgelöst wird, wenn Bit 0 = 1 und Bit 1 = 0 ist. Diese beiden Bit haben in beiden Kontrollregistern die gleiche Bedeutung.

In Abbildung 10.6 ist die Wirkung eines Impulses auf der Leitung CA2 gezeigt. Sind Bit 3, 4, 5 gleich Null, so löst weder eine positive noch

eine negative Flanke einen Interrupt aus. Auch diese 3 Bit haben in beiden Registern die gleiche Bedeutung.

CRA (CRB)

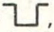
Bit 5	Bit 4	Bit 3	aktive Flanke an CA2 (CB2)	$\overline{IRQA}$ ($\overline{IRQB}$)
0	0	0		Kein Interrupt
0	0	1		Bit 6 = 0 Interrupt
0	1	0		Kein Interrupt
0	1	1		Bit 6 = 0 Interrupt

10.6 Interruptsteuerung CA2 (CB2)

Wird Bit 5 gleich Eins gesetzt, so haben die Steuerleitungen unterschiedliches Verhalten.
Über das Kontrollregister CRA erfolgt die Betriebsart Handshake Lesen an Tor A.

Mit Bit 3, 4 = 0 und Bit 5 = 1 ist CA2 eine Ausgangsleitung. Ein Interrupt auf CA1 setzt die CA2 Leitung auf 1. Diese wird erst dann wieder Null, wenn ein LOAD von Tor A - Befehl erfolgt ist. Mit der nächsten Bitbelegung wird auf CA2 nach einen LOAD von Tor A - Befehl ein negativer Impuls für die Dauer eines Maschinenzykluses ausgegeben. Mit den letzten beiden Bitbelegungen kann CA2 auf 0 oder 1 gesetzt werden.

CRA

Bit 5	Bit 4	Bit 3	Betriebsart	Beschreibung
1	0	0	Handshake Lesen	CA2 = 1 wenn Interrupt an CA1, CA2 = 0 bei LOAD-Befehl
1	0	1	Ausgangs-impuls	CA2 = , nach LOAD-Befehl
1	1	0		CA2 = 0
1	1	1		CA2 = 1

10.7 Betriebsart Handshake Lesen

Das Kontrollregister CRB bestimmt die Betriebsart Handshake schreiben an Tor B.

Mit Bit 5 = 1 und Bit 3 und 4 = 0 wird CB2 bei einem STORE an Tor B - Befehl auf Null gesetzt. Die Rücksetzung erfolgt dann durch einen aktiven Interrupt an CB1. Mit der Bitbelegung 101 wird nach jedem STORE an das Tor B - Befehl ein negativer Impuls auf CB2 ausgegeben.

Die Impulsdauer beträgt wiederum einen Maschinenzyklus. Durch die beiden anderen Bitmuster kann CB2 auf Null oder Eins gesetzt werden.

CRB

Bit 5	Bit 4	Bit 2	Betriebsart	Beschreibung
1	0	0	Handshake Schreiben	CB2 = 0 bei STORE-Befehl, CB2 = 1 bei Interrupt auf CB1
1	0	1	Ausgangs-impuls	CB2 = ‾|_|‾ , nach STORE-Befehl
1	1	0		CB2 = 0
1	1	1		CB2 = 1

10.8 Betriebsart Handshake Schreiben

Der 6520 eignet sich also vorzüglich zum Datentransfer zwischen Prozessor und peripheren Geräten.

Beim ATARI-Computer hat dieser Baustein die Adressen:

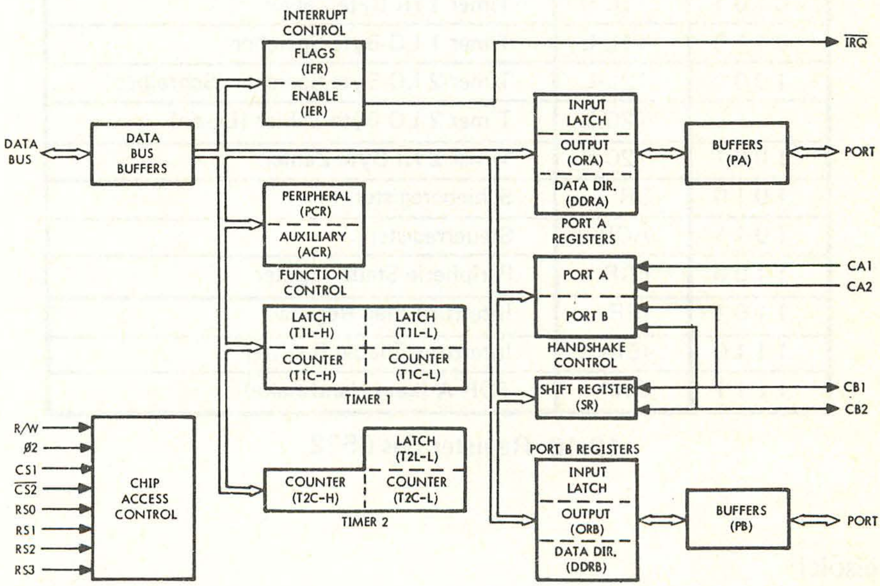
	HEX	DEZ
TORA / DDRA	D300	54016
TORB / DDRB	D301	54017
CRA	D302	54018
CRB	D303	54019

10.2 Der Baustein VIA 6522

Der VIA (Versatile Interface Adapter) 6522 enthält 2 bidirektionale Tore A und B, 2 16-bit Timer, ein Schieberegister und Steuerleitungen für den Handshake-Betrieb mit anderen Geräten.

Das Blockschaltbild des 6522 gleicht dem Blockschaltbild des 6520 (Abbildung 10.1). Der interne Aufbau des 6522 ist jedoch wesentlich komplizierter (Abbildung 10.9). Alle Funktionen werden über 16 adressierbare Register angewählt. Die Adressen dieser Register und ihre Bezeichnungen sind in Abbildung 10.10 angegeben.

Die Basisadresse des Bausteins hängt von dem ihm zugeordneten freien Speicherplatz im Rechner ab.



10.9 Interner Aufbau des 6522

10.21 Programmierung der Tore

Die 16 I/O-Leitungen können wahlweise als Eingangs- oder Ausgangsleitungen verwendet werden. Die Richtung wird über die Datenrich-

tungsregister DDRA und DDRB festgelegt. Wird in diesen Registern ein Bit = 1 gesetzt, so ist die entsprechende Leitung ein Ausgang, wird ein Bit = 0 gesetzt, so ist diese Leitung ein Eingang. Nach dem Einschalten und nach jedem RESET (BREAK) des Rechners sind alle Leitungen als Eingangsleitungen geschaltet.

Adresse	Name	Bezeichnung
0000	ORB	Ein- und Ausgangstor B
0001	ORA	Ein- und Ausgangstor A
0010	DDRB	Datenrichtungsregister B
0011	DDRA	Datenrichtungsregister A
0100	T1L-L	Timer 1 LO-Byte-Speicher (Schreiben)
	T1C-L	Timer 1 LO-Byte-Zähler (Lesen)
0101	T1C-H	Timer 1 HI-Byte-Zähler
0110	T1L-L	Timer 1 LO-Byte-Speicher
1000	T2L-L	Timer 2 LO-Byte-Speicher (Schreiben)
	T2C-L	Timer 2 LO-Byte-Zähler (Lesen)
1001	T2C-H	Timer 2 HI-Byte-Zähler
1010	SR	Schieberegister
1011	ACR	Steuerregister
1100	PCR	Peripherie Steuerregister
1101	IFR	Interrupt Flag Register
1110	IER	Interrupt Enable Register
1111	ORA	TOR A (kein Handshake)

10.10 Register des 6522

Beispiel:
Bit 0 bis 6 von Tor A sind Ausgänge, Bit 7 ein Eingang.

```
LDA #$7F
STA  DDRA
```

Die Festlegung der Datenrichtung braucht in einem Programm nur einmal gemacht werden. Sie bleibt bis zu einem RESET erhalten.

10.22 Programmierung der Timer

Der Timer 1 besteht aus 2 8-bit Speichern T1L–L und T1L H und einem 16-bit Zähler T1C–L und T1C–H. Die Zählerzelle T1C–L hat beim Schreiben und Lesen unterschiedliche Funktionen. Das Schreiben nach T1C–L entspricht einem Schreiben nach T1L–L, sie verhält sich wie eine Speicherzelle, beim Lesen von T1C–L wird das LO-Byte des Zählers ausgegeben. Ein Schreibbefehl nach T1C–H startet den Zähler. Dabei wird der Inhalt von T1L–L nach T1C–L übernommen. Der Inhalt des Zählers wird mit dem Maschinentakt $\emptyset_2$ erniedrigt. Wird der Zählerstand 0 erreicht, wird je nach programmierter Betriebsart ein Interrupt augelöst oder ein Signal an PB7 (Port B, Bit7) ausgegeben. Dabei kann der Inhalt von T1L–L und T1L–H wieder in den Zähler übernommen und somit ein fortlaufendes Signal erzeugt werden.

Die Betriebsart wird über Bit 6 und Bit 7 des Steuerregisters ACR bestimmt (Abbildung 10.11). Mit ACR6=1 und ACR7=1 wird die Betriebsart "free running mode" eingestellt. Bei jedem Nulldurchgang des Zählers ändert sich die Polarität des Signals an PB7. Dort entsteht ein Rechtecksignal mit der programmierten Zeit als halbe Periodendauer.

ACR7	ACR6	Betriebsart
0	0	Monoflop, nur Interrupt, kein Signal an PB7
0	1	Fortlaufende Interrupts, kein Signal an PB7
1	0	Monoflop, Interrupt, negativer Impuls an PB7
1	1	Fortlaufende Interrupts, Rechtecksignal an PB7

10.11 Festlegung der Timer Betriebsart

Abbildung 10.12 ist ein Programm zur Erzeugung einer Taktfrequenz von 100.0 ms.

Die in den Zähler eingegebene Zahl $C74E entspricht Dezimal 51023 und nicht dem theoretischen Wert von 50000. Dies ist durch die Taktfrequenz der Rechner bedingt, die nicht genau 1 MHz ist, sondern meist etwas darüber liegt.

```
0800                ;
0800                ;
0800                ;******************************
0800                ;*                            *
0800                ;* RECHTECKGENERATOR MIT      *
0800                ;* 100.0 MS PERIODENDAUER     *
0800                ;*                            *
0800                ;******************************
0800                ;
0800                ;
0800                ACR     EQU $C0CB
0800                T1CL    EQU $C0C4
0800                T1CH    EQU $C0C5
0800                MONITO  EQU $FF59
0800                ;
0800                ;
0800 A9C0                   LDA #$C0          ; BETRIEBSART SETZEN
0802 8DCBC0                 STA ACR
0805 A94E                   LDA #$4E          ; LO BYTE LADEN
0807 8DC4C0                 STA T1CL
080A A9C4                   LDA #$C4          ; HI BYTE LADEN
080C 8DC5C0                 STA T1CH          ; UND TIMER STARTEN
080F 4C59FF                 JMP MONITO
0812                ;
0812                ;
                           END
```

10.12 Rechteckgenerator

Mit dem Befehl STA T1CH wird der Timer gestartet und läuft dann unabhängig von der CPU. An dieses Programmteil kann nun ein völlig anderes Programm angeschlossen werden, ohne die Funktion des 6522 Timers zu unterbrechen. Die Ausgabe des Rechtecks wird in einem Programm nur durch eine Änderung der Betriebsart in ACR beendet.

Soll eine Frequenzänderung durchgeführt werden, so wird die neue Zeit in die Speicherzellen T1L–L und T1L–H geschrieben. Diese Zahl wird dann beim nächsten Nulldurchgang in den Zähler übernommen und somit die Frequenz verändert.

Der Timer 2 kann wie der Timer 1 als Monoflop betrieben werden.

Darüberhinaus kann er als Zähler für negative Flanken externer Geräte an PB6 eingesetzt werden. Seine Betriebsart wird durch Bit 5 von ACR bestimmt.

ACR5	= 0	Monoflop
ACR5	= 1	Zählen negativer Flanken an PB6.

Verbindet man PB7 und PB6 durch eine Drahtbrücke, so erhält man so eine Stoppuhr zur Messung von Programmlaufzeiten oder einen Taktgeber für Prozess-Steuerungen.

10.23 Programmierung des Schieberegisters

Das Register SR ist ein Schieberegister, das entweder parallele Daten von Prozessor seriell an externe Geräte ausgeben oder serielle Daten von externen Geräten aufnehmen kann. Dazu kann ein externer Takt, ein Takt von Timer 2 oder der Maschinentakt verwendet werden. Die Betriebsart wird durch die BIT 4, 3, 2 vom Steuerregister ACR festgelegt. Ihre Bedeutung zeigt Abbildung 10.13.

ACR4	ACR3	ACR2	
0	0	0	Schieberegister abeschaltet
0	0	1	Einlesen mit Takt von Timer 2
0	1	0	Einlesen mit 0_2 Takt
0	1	1	Einlesen mit externen Takt
1	0	0	Auslesen "free running mode" mit Takt von Timer 2
1	0	1	Auslesen mit Takt von Timer 2
1	1	0	Auslesen mit 0_2 Takt
1	1	1	Auslesen mit externen Takt

10.13 Festlegung der SR Betriebsart

Der Inhalt des Schieberegisters wird über die Steuerleitung CB2 ein- oder ausgegeben. Erfolgt dies unter Verwendung eines externen Taktes, so wird dieser über CB1 eingegeben. Der interne Takt wird ebenfalls zur Synchronisierung externer Geräte über CB1 ausgegeben.

Wird der Takt vom TIMER2 vorgegeben, so ist er in Verbindung mit dem Schieberegister nur ein 8-bit Zähler, so daß die langsamste Schiebefrequenz ca. 0.5 ms beträgt, da das Ein- oder Auslesen des Schieberegisters bei jedem 2. Nulldurchgang des Zählers stattfindet.

Beispiel:
Freilaufender Rechteckgenerator mit änderbaren Tastverhältnis (Abbildung 10.14).

```
0800           ;
0800           ;
0800           ;*****************************
0800           ;*                           *
0800           ;*  RECHTECKGENERATOR MIT    *
0800           ;*  AENDERBAREM TASTVERHAELT-*
0800           ;*  NISS                     *
0800           ;*                           *
0800           ;*****************************
0800           ;
0800           ;
0800            ACR     EQU $COCB
0800            T2LL    EQU $COC8
0800            SR      EQU $COCA
0800            MONITO  EQU $FF59
0800           ;
0800 A9FF               LDA #$FF        ; TIMER 2 AUF LANGSAMSTE
0802 8DC8CO             STA T2LL        ; FREQUENZ STELLEN
0805 A910               LDA #$10        ; BETRIEBSART WAEHLEN
0807 8DCBCO             STA ACR
080A A90F               LDA #$OF        ; 4 X DIE 0 UND 4 X DIE 1
080C 8DCACO             STA SR          ; AUSGEBEN
080F 4C59FF             JMP MONITO
0812           ;
0812           ;
                        END
```

10.14 Rechteck 2

Durch Ändern der Bitbelegung des Schieberegisters SR lässt sich das Tastverhältnis, durch Ändern von T2L–L lässt sich die Frequenz des Rechteckgenerators verändern.

Mit den beiden verbleibenden Bit des Steuerregisters ACR lassen sich Bitmuster an Tor A und Tor B speichern, wenn diese als Eingänge geschaltet sind.

Ist Bit 0 vom ACR gleich Null, so wird bei einem LDA TORA das augenblickliche Bitmuster von Tor A in den Rechner übernommen.

Wird Bit 0 auf Eins gesetzt, so wird das anliegende Bitmuster solange gespeichert, bis ein LDA TORA erfolgt. Änderungen des Bitmusters an den Eingängen hat keinen Einfluß auf das gespeicherte Bitmuster. Das Einschreiben des Bitmusters erfolgt mit dem Setzen des CA1

Interruptmerkers im IFR Register (siehe 10.16). Die Beduetung der einzelnen Bit des Steuerregisters ACR zeigt Abbildung 10.15.

BIT	7	6	5	4	3	2	1	0
ACR	TIMER1		TIM– ER2	SCHIEBEREG.			TORA SPE	TORB CHERN

B

10.15 Bitbelegung ACR

Eine Überwachung der Timer und des Schieberegister findet im IFR (Interrupt Flag Register) statt. Die Bitbelegung zeigt Abbildung 10.16.

BIT	7	6	5	4	3	2	1	0
IFR	IRQ	T1	T2	CB1	CB2	SR	CA1	CA2

10.16 Bitbelegung IFR

Bei einer Schreib- oder Leseoperation mit dem Register SR wird Bit 2 von IFR gesetzt, und nach 8 Schiebetakten zurückgesetzt. Vom Programm aus muß also die Zahl der Schiebebefehle nicht mitgezählt werden.

Auf die gleiche Weise können die beiden Timer überwacht werden. Für sie sind Bit 5 (Timer 2) und Bit 6 (Timer 1) zuständig. Sie werden beim Starten des Zählers (Schiebebefehl nach T1C–H, bzw. T2C–H) gesetzt und beim Nulldurchgang des Zählers wieder zurück gesetzt. Dies wird im Programm MONOFLOP (Abbildung 10.17) benutzt. Beim Abfragen der Bit ist zu beachten, daß diese "Active low" sind, daß das gesetzte Bit gleich Null ist.

10.24 Datenübertragung mit externen Geräten

Die Bit 0, 1, 3 und 4 des IFR werden zur Steuerung der Datenübertragung mit externen Geräten benötigt. Das Setzen und Rücksetzen

dieser Bit erfolgt durch Programmierung des PCR (Peripheral Control Register). Die Bitbelegung zeigt Abbildung 10.18.

```
0800          ;
0800          ;
0800          ;****************************
0800          ;*                          *
0800          ;*        MONOFLOP          *
0800          ;*                          *
0800          ;****************************
0800          ;
0800          ;
0800          ACR      EQU $COCB
0800          T1CL     EQU $COC4
0800          T1CH     EQU $COC5
0800          IFR      EQU $COCD
0800          ;
0800 A980     MOMOFL   LDA #$80     ; BETRIEBSART MONOFLOP
0802 8DCBCO            STA ACR      ; SETZEN
0805 A950              LDA #$50
0807 8DC4CO            STA T1CL     ; ZEIT SETZEN
080A A9C7              LDA #$C7
080C 8DC5CO            STA T1CH     ; IFR6=0 TIMER STARTEN
080F ADCDCO   M        LDA IFR
0812 2940              AND #$40     ; IFR6 AUSBLENDEN
0814 FOF9              BEQ M
0816 60                RTS          ; RUECKKEHR MIT IFR6=1
0817          ;
0817          ;
                       END
```

10.17 Monoflop

BIT	7	6	5	4	3	2	1	0
PCR	CB2			CB1	CA2			CA1

10.18 Bitbelegung PCR

Bit 0 von PCR bestimmt, mit welcher Flanke an der Eingangsleitung CA1 das Bit 1 in IFR gesetzt wird. Ist PCR0 = 0 so wird CA1 durch eine negative Flanke, mit PCR0 = 1 durch eine positive Flanke gesetzt. Die Rücksetzung erfolgt durch eine Schreib- oder Leseoperation auf Tor A. Auf die gleiche Weise kann über PCR4 das Setzen des Bits IFR4 programmiert werden. Das Rücksetzen erfolgt durch eine Schreib- oder Leseoperation mit Tor B.

Für das Setzen oder Rücksetzen von CA2 bzw. CB2 bestehen insgesamt 8 Möglichkeiten. Diese sind in Abbildung 10.19 zusammgestellt.

PCR3	PCR2	PCR1	
0	0	0	IFR0 wird durch eine negative Flanke an CA2 gesetzt, Rücksetzen durch Lesen oder Schreiben von TORA
0	0	1	Interrupt - Betrieb. Negative Flanke setzt IFR0, kein Rücksetzen durch Lesen oder Schreiben von TORA
0	1	0	IFR0 wird durch eine positive Flanke gesetzt, Rücksetzen durch Lesen oder Schreiben von TORA
0	1	1	Interrupt - Betrieb. Positive Flanke setzt IFR0, kein Rücksetzen durch Lesen oder Schreiben von TORA
1	0	0	Handshake - Betrieb. CA2 Ausgang wird nach Schreiben oder Lesen von TORA auf 0 gesetzt. Rücksetzen durch Impuls an CA1
1	0	1	Impuls - Betrieb. Nach Lesen oder Schreiben von TORA wird CA2 einen Takt lang 0
1	1	0	Setzt CA2 auf 0
1	1	1	Setzt CA2 auf 1

10.19 Programmierung von CA2, CB2

Mit den Bit PCR3, 2, 1 wird CA2, mit den Bit PCR7, 6, 5 wird CB2 programmiert.

10.25 Interrupt-Verarbeitung

Dem IFR-Register ist ein zweites Register, das IER (Interrupt Enable Register) zugeordnet.

Den Bit 0 bis 6 des IFR entsprechen die Bit 0 bis 6 des IER. Ist das entsprechende Bit des IER gleich 1, so wird der Interrupt an den Prozessor weitergeleitet. Bit 7 des Bitmusters, das in das IER zu schreiben ist, bestimmt, ob die Interrupt-Enable-Bits gesetzt oder gelöscht

werden sollen. Ist Bit 7 gleich 1, so werden alle Bit 0 bis 6 gesetzt, die im Bitmuster eine 1 enthalten, eine 0 im Bitmuster ändert das entsprechende Interrupt-Enable-Bit nicht.
Beispiel: Interrupt von Timer 1 zulassen

```
LDA  #% 10100000
STA  IFR ; Interrupt löschen
STA  IER ; Interrupt einschalten
```

Soll im IER ein Bit gelöscht werden, und damit eine Programmunterbrechung von diesem Teil des 6522 nicht mehr zugelassen werden, so ist Bit 7 des Bitmusters 0. Alle Bit 0 bis 6, die im Bitmuster 1 sind, löschen das entsprechende Interrupt-Enable-Bit. Eine Null im Bitmuster lässt dieses ungeändert.
Beispiel: Interrupt von Timer 1 abschalten

```
LDA  #% 00100000
STA  IFR ; Interrupt löschen
STA  IER ; Interrupt abschalten
```

Da für die 6 Möglichkeiten eines Interrupts nur eine IRQ Leitung zum Prozessor zur Verfüngung steht, muß durch Ausblenden des IFR-Registers in dem Interrupt Service Programm festgestellt werden, welchen Teil des 6522 den Interrupt ausgelöst hat.

Der 6502 Prozessor erkennt zwei verschiedene Interrupts. Den NMI (Non Maskable Interrupt) und den IRQ (Interrupt Request).

Der Unterschied zwischen beiden besteht darin, daß der NMI nicht abgeschaltet werden kann, während der IRQ durch Setzen oder Löschen des Interrupt Bits im Statusregister zugelassen oder nicht zugelassen wird. Durch SEI wird ein Interrupt über IRQ abgeschaltet, durch CLI wird er zugelassen.

Wird eine dieser beiden Leitungen physikalisch auf Null gesetzt, so wird das laufende Programm nach Beendigung des gerade auszuführenden Befehls unterbrochen und ein indirekter Sprung über eine festgelegte Speicherzelle ausgeführt.

Für den NMI sind dies die Zellen $FFFA und $FFFB und für den IRQ die Zellen $FFFE und $FFFF. Da diese Zellen ROM-Zellen sind, und somit nicht durch Programm geändert werden können, muß man die weitere Behandlung des Interrupts über die in diesen Zellen programmierten Adressen verfolgen.

Beim APPLE II ist für den NMI die Adresse $3BFB programmiert. Dort findet man:

$03FB – 4C 65 FF

Dies ist ein Rücksprung in den Monitor. In diese beiden RAM-Zellen kann nun ein Sprung in ein eigenes Interruptbehandlungs-Programm programmiert werden.

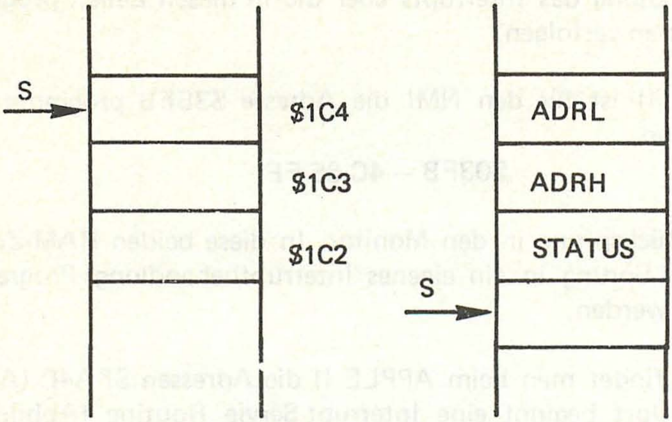
Für den IRQ findet man beim APPLE II die Adressen $FA40 (Autostart ROM). Dort beginnt eine Interrupt-Servie Routine (Abbildung 10.20). Bei einem Interrupt wird neben dem augenblicklichen Stand des Befehlszählers auch noch der Prozessorstatus im Stapel abgelegt (Abbildung 10.21).

```
*
FA40-   85 45      STA   $45        ; AKKUMULATOR RETTEN
FA42-   68         PLA              ; INHALT VOM STAPEL HOLEN
FA43-   48         PHA              ; UND WIEDER ZURÜCKSCHREIBEN
FA44-   0A         ASL
FA45-   0A         ASL
FA46-   0A         ASL              ; 3 MAL LINKS SCHIEBEN
FA47-   30 03      BMI   $FA4C      ; WENN NEGATIV, DANN WAR ES BRK
FA49-   6C FE 03   JMP   ($03FE)    ; SONST ZUR INTERRUPT–SERVICE–ROUTINE
FA4C-   28         PLP              ; BEGINN BRK–PROGRAMM
FA4D-   20 4C FF   JSR   $FF4C
FA50-   68         PLA
FA51-   85 3A      STA   $3A
FA53-   68         PLA
```

10.20 IRQ-Routine Apple II

In der Interrupt-Service-Routine wird zuerst der Akkumulator gerettet und dann der Prozessorstatus in den Akkumulator geholt. Durch gleichzeitiges Zurückschreiben wird der Stapelzeiger nicht verändert. Wenn dieser Interrupt durch einen BRK-Befehl (Software-Interrupt) ausgelöst wurde, ist im Statusregister das BRK-Bit gesetzt. Dies wird durch 3 maliges Linksschieben des Akkumulatorinhaltes festgestellt. War es ein BRK-Befehl, so wird mit BMI in den BRK-Programmteil verzweigt. Wenn nicht, folgt ein indirekter Sprung über die Speicher-

zellen $3FE, $3FF zurück in den Monitor. In diesen beiden Zellen kann der Benutzer den Beginn einer eigenen IRQ Service Routine programmieren.



10.21 **Stapelbelegung bei einem Interrupt**

Die Rückkehr in das ursprüngliche Programm erfolgt mit RTI (Return from Interrupt). Dabei wird der Prozessorstatus zurückgeschrieben und die Adresse des nächsten Befehls vom Stapel in den Programmzähler übernommen.

10.26 Druckersteuerung mit einem 6522

Ein Drucker mit einer 8-bit Parallel-Schnittstelle soll über einen 6522 an ein 6502 System angeschlossen werden.

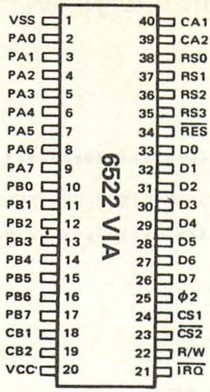
Die ASCII-Zeichen werden über das Tor A an den Drucker übergeben. Die beiden Steuersignale STR und $\overline{\text{ACK}}$ werden über das Tor B ausgegeben bzw. aufgenommen.

Der Druckbeginn wird durch einen positiven Impuls STR eingeleitet. Als Leitung zum Drucker wird dafür PB0 verwendet. Dieser antwortet auf STR mit $\overline{\text{ACK}}$. Dieses Signal ist solange Null, bis der Drucker bereit ist, ein neues Zeichen zu übernehmen. Mit PB7 wird diese

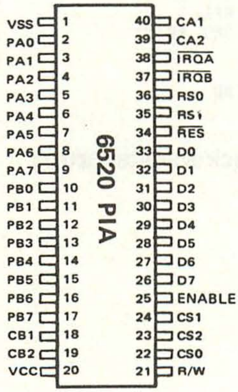
Leitung überwacht. Damit erhält man folgendes Programm (Abbildung 10.22).

```
0800             ;
0800             ;
0800             ;******************************
0800             ;*                            *
0800             ;* ANSTEUERUNG EINES DRUCKERS*
0800             ;*                            *
0800             ;******************************
0800             ;
0800             ;
0800             TORA    EQU $C091
0800             TORB    EQU $C090
0800             DDRA    EQU $C093
0800             DDRB    EQU $C092
0800             ;
0800             GETCH   EQU $FD00
0800             ;
0800 A9FF        PRINT   LDA #$FF
0802 8D93C0              STA DDRA
0805 A903                LDA #$03
0807 8D92C0              STA DDRB
080A A902                LDA #$02                  ; 2 --> TORB
080C 8D90C0              STA TORB
080F 2000FD              JSR GETCH
0812 8D91C0              STA TORA                  ; ASCII --> DRUCKER
0815 4E90C0              LSR TORB                  ;
0818 0E90C0              ASL TORB                  ; STR --> DRUCKER
081B AD90C0      P       LDA TORB                  ; WARTEN AUF ACK
081E 10FB                BPL P
0820 30DE                BMI PRINT                 ; NEUES ZEICHEN
0822             ;
0822             ;
                         END
```

10.22 Druckeransteuerung



Pinbelegung 6522



Pinbelegung 6520

Fehlersuche in Programmen

11. Fehlersuche in Programmen

In den wenigsten Fällen wird ein Programm sofort fehlerfrei laufen. Wenn dies der Fall ist, so sollte man trotzdem skeptisch sein, denn nicht alle Schleifen werden immer sofort durchlaufen. Auf der anderen Seite sollte man unerklärliche Fehler nicht auf Hardware-Fehler zurückführen.

In 99,9 Prozent der Fälle ist dies nicht der Fall . Die Fehlersuche sollte schon beim Programmieren berücksichtigt werden. Kleinere Programmblöcke und kurze Unterprogramme sind leicht zu testen. Dazu lohnt es sich oft, ein kleines Programm zu schreiben, das wie das spätere fertige Programm, die Daten an das zu prüfende Programm übergibt. Das gewünschte Ergebnis wird durch Anzeigen bestimmter Zelleninhalte überprüft.

In den meisten Fällen wird sich aber folgendes ereignen. Man startet das Programm, und nichts passiert. Es erscheint kein erwarteter Bildschirmausdruck, keine Signale werden ausgelöst, denn das Programm ist "versoffen", und kann nur durch RESET angehalten werden. Was ist nun zu tun. Als erstes sollte das Programm ausgedruckt werden und mit den vorhergemachten Aufzeichnungen verglichen werden. Denn einer der häufigsten Fehler ist, beim Schreiben eines Programms, das Vergessen einer Assemblerzeile.

Wenn dabei nichts entdeckt wird, muß man durch Programmänderungen versuchen den Fehler einzukreisen. Dabei gilt ein eiserner Grundsatz:

Für jeden Testlauf nur **eine einzige** Änderung einführen.

Diese Änderung muß auch im Programmausdruck vermerkt werden. Für diese Programmänderung verwendet man am besten den BRK-Befehl. Jeder anständige Monitor hat eine BREAK-Routine, bei welcher der Inhalt des Akkumulators, des Indexregister, des Stapelzeigers und des Statusregisters auf den Bildschirm ausgegeben wird.

Eine unendliche Schleife lässt sich dann auf folgende Weise finden. Man ändert den ersten Befehl nach einem JSR in einen BRK Befehl um und stellt so fest, ob das Unterprogramm verlassen wird. Ist dies der Fall, wird die Änderung rückgängig gemacht und ein BRK nach dem nächsten Unterprogrammsprung eingeführt. Wird das Unterprogramm nicht velassen, so muß der Fehler auf die gleiche Weise dort gesucht werden.

Beliebte Fehler, die zu unendlichen Schleifen führen können sind folgende zwei Beispiele:

```
MO  LDX #$00
    LDA PLATZ,X
    INX
    CPX #$C0
    BNE MO
```

und

```
    INC ADRL
    CMP #$A0
    BNE WOHIN
```

Der Leser kann sich überlegen, was an diesen Beispielen falsch ist.
Einzelne Monitore verfügen über TRACE-Programme. Damit kann ein Programmteil in einzelnen Schritten ausgeführt werden. Nach jedem Schritt wird der Inhalt der Register angezeigt.

Bei der Papier- und Bleistiftmethode ist der häufigste Fehler ein falsch berechneter Sprung. Diesem kann man durch dieses Einzelschitt-verfahren oder durch Disassemblierung auf die Spur kommen.

Ist der Fehler erkannt, so kann er beseitigt werden. Dies ist bei Assemblern mit Texteditoren einfach. Die neuen Zeilen werden eingefügt oder geändert und das ganze neu übersetzt. Bei Miniassemblern ist es oft einfacher, einen falschen Befehl herauszunehmen und durch

einen Sprung an das Ende des Programms zu ersetzen. Dort wird die richtige Befehlsfolge abgearbeitet und dann an die weiterführende Stelle zurückgesprungen. Das ergibt zwar keine übersichtlichen Programme, aber es ist oft der einzige Ausweg um einen Fehler zu beseitigen. Es ist deshalb vorteilhaft, beim Arbeiten mit Miniassemblern, nach einigen Programmzeilen durch Einfügen von mehreren NOP (mindestens 3) Luft für Änderungen zu lassen.

Beim Fehlersuchen mit dem BRK-Befehl ist noch folgendes zu beachten. Der BRK ist ein Software-Interrupt. Der Programmzähler zeigt danach auf das 2. Byte nach dem BRK. Kann das Programm durch einen Monitor-Befehl weiter ausgeführt werden, so ist ein BRK gefolgt von 2 NOP's einzufügen.

Eine weitere Möglichkeit, einen Programmablauf zu verfolgen, ist die Ausgabe von Zeichen im Programmablauf. An bestimmten Stellen im Programm wird ein

```
LDA  ZEICHEN
JSR  OUTCHR
```

eingefügt.

Damit kann man am Bildschirm verfolgen, welche Programmteile durchlaufen werden, oder welche nicht durchlaufen werden. Der Zeitaufwand für die Fehlersuche übertrifft oft bei weitem die Zeit, die man benötigt hat, um ein Programm zu schreiben. Ist der Fehler nicht zu finden, ist die Neuprogrammierung der schnellste Weg. Im übrigen gilt der Spruch:

"Der Computer hat immer recht."

NOTIZEN

Anhang

Anhang

Im folgenden Anhang sind nochmals alle Befehle in ihrer Assembler-Schreibweise, als Hexcode und mit der benötigten Byte-Zahl angegeben. Dazu kommt noch die Befehlsausführungszeit mit der Angabe der benötigten Maschinenzyklen.

ADC

A+M+C → A

	OP	NB	NC
ADC #$AA	69	2	2
ADC $1000	6D	3	4
ADC $1000,X	7D	3	4
ADC $1000,Y	79	3	4
ADC $10	65	2	3
ADC $10,X	75	2	4
ADC ($10,X)	61	2	6
ADC ($10),Y	71	2	5

AND

A ∧ M → A

	OP	NB	NC
AND #$AA	29	2	2
AND $1000	2D	3	4
AND $1000,X	3D	3	4
AND $1000,Y	39	3	4
AND $10	25	2	3
AND $10,X	35	2	4
AND ($10,X)	21	2	6
AND ($10),Y	31	2	5

ASL

Siehe Bild 3.2

	OP	NB	NC
ASL $1000	0E	3	6
ASL $100,X	1E	3	7
ASL $10	06	2	5
ASL $10,X	16	2	6
ASL	0A	1	2

BCC

BRANCH ON C=0

	OP	NB	NC
BCC $2F	90	2	2

BCS

BRANCH ON C=1

	OP	NB	NC
BCS $2F	B0	2	2

BEQ

BRANCH ON Z=1

BEQ $2F

OP	NB	NC
F0	2	2

BIT

A∧M

BIT $1000
BIT $10

OP	NB	NC
2C	3	4
24	2	3

BMI

BRANCH ON N=1

BMI $2F

OP	NB	NC
30	2	2

BNE

BRANCH ON Z=0

BNE $2F

OP	NB	NC
D0	2	2

BPL

BRANCH ON N=0

BPL $2F

OP	NB	NC
10	2	2

BRK

BREAK

BRK

OP	NB	NC
00	1	7

BVC

BRANCH ON V=0

BVC $2F

OP	NB	NC
50	2	2

BVS

BRANCH ON V=1

BVS $2F

OP	NB	NC
70	2	2

CLC

C=0

CLC

OP	NB	NC
18	1	2

CLD

D=0

CLD

OP	NB	NC
D8	1	2

CLI

I=0

CLI

OP	NB	NC
58	1	2

CLV

V=0

CLV

OP	NB	NC
B8	1	2

CMP

A–M

```
CMP #$AA
CMP $1000
CMP $1000,X
CMP $1000,Y
CMP $10
CMP ($10,X)
CMP ($10),Y
```

OP	NB	NC
C9	2	2
CD	3	4
DD	3	4
D9	3	4
C5	2	3
C1	2	6
D1	2	5

CPX

X–M

```
CPX #$AA
CPX $1000
CPX $10
```

OP	NB	NC
E0	2	2
EC	3	4
E4	2	3

CPY

Y–M

```
CPY #$AA
CPY $1000
CPY $10
```

OP	NB	NC
C0	2	2
CC	3	4
C4	3	4

DEC

M–1 → M

DEC $1000
DEC $1000,X
DEC $10
DEC $10,X

OP	NB	NC
CE	3	6
DE	3	7
C6	2	5
D6	2	6

DEX

X–1 → X

DEX

OP	NB	NC
CA	1	2

DEY

Y–1 → Y

DEY

OP	NB	NC
88	1	2

EOR

A ⊻ M → A

	OP	NB	NC
EOR #$AA	49	2	2
EOR $1000	4D	3	4
EOR $1000,X	5D	3	4
EOR $1000,Y	59	3	4
EOR $10	45	2	3
EOR $10,X	55	2	4
EOR ($10,X)	41	2	6
EOR ($10),Y	51	2	5

INC

M+1 → M

	OP	NB	NC
INC $1000	EE	3	6
INC $1000,X	FE	3	7
INC $10	E6	2	5
INC $10,X	F6	2	6

INX

X+1 → X

INX

OP	NB	NC
E8	1	2

INY

Y+1 → Y

INY

OP	NB	NC
C8	1	2

JMP

JMP $1000
JMP ($1000)

OP	NB	NC
4C	3	3
6C	3	5

JSR

	OP	NB	NC
JSR $1000	20	3	6

LDA

M → A

	OP	NB	NC
LDA #$AA	A9	2	2
LDA $1000	AD	3	4
LDA $1000,X	BD	3	4
LDA $1000,Y	B9	3	4
LDA $10	A5	2	3
LDA $10,X	B5	2	4
LDA ($10,X)	A1	2	6
LDA ($10),Y	B1	2	5

LDX

M → X

	OP	NB	NC
LDX #$AA	A2	2	2
LDX $1000	AE	3	4
LDX $1000,Y	BE	3	4
LDX $10	A6	2	3
LDX $10,Y	B6	2	4

LDY

M → Y

Befehl	OP	NB	NC
LDY #$AA	A0	2	2
LDY $1000	AC	3	4
LDY $1000,X	BC	3	4
LDY $10	A4	2	3
LDY $10,X	B4	2	4

LSR

Siehe Bild 3.3

Befehl	OP	NB	NC
LSR $1000	4E	3	6
LSR $1000,X	5E	3	7
LSR $10	46	2	5
LSR $10,X	56	2	6
LSR	4A	1	2

NOP

NO OPER

Befehl	OP	NB	NC
NOP	EA	1	2

ORA

A V M → A

	OP	NB	NC
ORA #$AA	09	2	2
ORA $1000	0D	3	4
ORA $1000,X	1D	3	4
ORA $1000,Y	19	3	4
ORA $10	05	2	3
ORA $10,X	15	2	4
ORA ($10,X)	01	2	6
ORA ($10),Y	11	2	5

PHA

A → Ms, S–1 → S

	OP	NB	NC
PHA	48	1	3

PHP

P → Ms, S–1 → S

	OP	NB	NC
PHP	08	1	3

PLA

S+1 → S, Ms → A

PLA

OP	NB	NC
68	1	4

PLP

S+1 → S, Ms → P

PLP

OP	NB	NC
28	1	4

ROL

Siehe Bild 3.4

	OP	NB	NC
ROL $1000	2E	3	6
ROL $1000,X	3E	3	7
ROL $10	26	2	5
ROL $10,X	36	2	6
ROL	2A	1	2

ROR

Siehe Bild 3.5

	OP	NB	NC
ROR $1000	6E	3	6
ROR $1000,X	7E	3	7
ROR $10	66	2	5
ROR $10,X	76	2	6
ROR	6A	1	2

RTI

RETURN F. INT

	OP	NB	NC
RTI	40	1	6

RTS

RETURN F. SUB

	OP	NB	NC
RTS	60	1	6

SBC

A–M–$\overline{C}$ → A

	OP	NB	NC
SBC #$AA	E9	2	2
SBC $1000	ED	3	4
SBC $1000,X	FD	3	4
SBC $1000,Y	F9	3	4
SBC $10	E5	2	3
SBC $10,X	F5	2	4
SBC ($10,X)	E1	2	6
SBC ($10),Y	F1	2	5

SEC

C=1

	OP	NB	NC
SEC	38	1	2

SED

D=1

	OP	NB	NC
SED	F8	1	2

SEI

I=1

	OP	NB	NC
SEI	78	1	2

STA

A → M

	OP	NB	NC
STA $1000	8D	3	4
STA $1000,X	9D	3	5
STA $1000,Y	99	3	5
STA $10	85	2	3
STA $10,X	95	2	4
STA ($10,X)	81	2	6
STA ($10),Y	91	2	6

STX

X → M

	OP	NB	NC
STX $1000	8E	3	4
STX $10	86	2	3
STX $10,Y	96	2	4

STY

Y → M

	OP	NB	NC
STY $1000	8C	3	4
STY $10	84	2	3
STY $10,X	94	2	4

TAX

A → X

	OP	NB	NC
TAX	AA	1	2

TAY

A → Y

	OP	NB	NC
TAY	A8	1	2

TSX

S → X

TSX

OP	NB	NC
BA	1	2

TXA

X → A

TXA

OP	NB	NC
8A	1	2

TXS

X → S

TXS

OP	NB	NC
9A	1	2

TYA

Y → A

TYA

OP	NB	NC
98	1	2

Einige Tabellen, die im Text verstreut sind, werden zur besseren Übersicht noch einmal aufgeführt: